AI Communications 26 (2013) 179–209
DOI 10.3233/AIC-130558
IOS Press

Automatic component abstraction for Model-Based Diagnosis on relational models

Gianluca Torta * and Pietro Torasso
Dipartimento di Informatica, Università di Torino, Torino, Italy
E-mails: {torta, torasso}@di.unito.it

Abstract. In the present paper, we address the problem of automatically synthesizing component abstractions by taking into account the level of observability of the system as well as restrictions on its operating conditions. Compared with previous work, the proposed approach can be applied to a significantly wider class of systems, namely those whose nominal and faulty behaviors can be modeled with finite-domain relations.

The computed abstractions are specifically tailored for the Model-Based Diagnosis task, with the main goal of getting fewer and more informative diagnoses through the use of abstract models. To this end, we define a spectrum of *indiscriminability* relations among the states of subsystems, and formally prove that respecting indiscriminability is both a necessary and sufficient condition for abstracting the original model without losing any relevant diagnostic information.

We present an algorithm for the computation of abstractions that implements two specially important cases of indiscriminability, namely local and global-indiscriminability. The implemented system is exploited to collect experimental results that confirm the benefits of using the abstractions for diagnosis, in terms of both the number of returned diagnoses and the computational cost.

Keywords: Component abstraction, task-dependent abstraction, Model-Based Diagnosis, relational models, minimum-cardinality diagnoses

1. Introduction

In the Model-Based Diagnosis (MBD) field, several works have investigated the possibility of performing the diagnostic reasoning on an abstract model instead of (or before) considering a detailed model involving a larger number of components and/or more refined behaviors for each component (see for example [2,7,13, 17,19,20,22,23,28,29]).

In *hierarchical* diagnosis [7,17,20,23], the system is modeled at two or more levels of abstraction and the abstract models are used to focus diagnostic reasoning and, therefore, improve the efficiency. In particular, once abstract diagnoses have been computed, they must be refined down to the ground level, where each of the abstract diagnoses may correspond to many detailed diagnoses; such a refinement is usually performed by exploiting the detailed system models.

Another way to exploit abstraction in MBD has been proposed in other works [22,28,29], which aim at automatically synthesizing abstract models that are simpler than the corresponding ground models but can completely replace them for diagnostic reasoning, without incurring any loss of relevant diagnostic information; following [22], we will refer to this kind of abstractions as *task-dependent* abstractions. In fact, the main goal of this approach to abstraction is to forget details of the model that are not relevant for a specific task (i.e., diagnosis) in a possibly restricted set of situations (defined, e.g., by the observability of the system, its operating conditions and the desired precision of diagnoses), thus tailoring the model to the use that will be made of it.

In this paper, we will present an approach which is able to automatically synthesize task-dependent component abstractions that are based on suitable notions of indiscriminability among system states (i.e., assignments of behavioral modes to system components), and ensure that diagnoses are fully preserved.

The two main reasons to use such abstractions instead of the ground model for diagnostic reasoning are:

- the computational cost is lower, since in general the abstract diagnosis search space is smaller than the ground one,

*Corresponding author: Gianluca Torta, Dipartimento di Informatica, Corso Svizzera 185, 10149 Torino, Italy. E-mail: torta@di.unito.it.

0921-7126/13/$27.50 © 2013 – IOS Press and the authors.

This article is published online with Open Access and distributed under the terms of the Creative Commons Attribution Non-Commercial License.

- the returned abstract diagnoses are fewer and more informative than the ground ones.

We will particularly stress the second point, since one of the main challenges of diagnostic reasoning is that all of the possible explanations of the observed system behavior must be computed. The number of such explanations can be exponential in the size of the system, and tends to be particularly high when the system model specifies too many details which are irrelevant given the specific conditions under which diagnosis is performed.

An alternative way to mitigate this problem, which is often adopted in MBD, is that of computing just the preferred diagnoses, instead of the whole set of diagnoses. A common criterion for establishing a preference order is minimum cardinality, i.e., the diagnoses with the minimum number of faults are preferred. We will show that our approach to abstraction can be easily exploited for the computation of minimum-cardinality diagnoses, and we will present experimental results that show how abstraction can yield significant benefits also when it is combined with the preference for minimum cardinality diagnoses.

Our approach can be applied to the class of systems that can be modeled as finite-domain relations which specify the nominal as well as one or more faulty behaviors for each component. This kind of models can capture non-directional and non-deterministic system behavior; in other words, relational models do not require that the system behavior can be expressed as a function from the system inputs to the system ouputs, as it is the case for simpler models such as those of combinatorial digital circuits. Typical examples of non-directional models are found in hydraulic systems and analog circuits, where the end points of a pipe or a resistor can act either as input or output, depending on the current direction of the flow of the fluid or electricity.

In order to automatically abstract system components, one obvious question concerns which sets of ground components should be abstracted, and in which order. However, when a system model specifies multiple behavioral modes, there is an additional fundamental question: how do we determine the behavioral modes of abstract components, i.e., how do we synthesize the behavior of an abstract component starting from the models of the ground components that it abstracts?

In this paper, we will propose an approach that addresses these questions based on sound theoretical grounds. First of all, we will briefly consider the state of the art, focusing in particular on the kind of relationship between abstract and ground diagnoses that is guaranteed by existing hierarchical approaches to model abstraction; this should clearly position the approach presented in this paper, and motivate its development.

After introducing formal definitions of system models, abstraction mappings, ground and abstract diagnoses, we will present a spectrum of notions of *indiscriminability* among mode assignments to system components that play a major role in defining the relation between the behavioral modes of an abstract component and the behavioral modes of its (ground) subcomponents. In particular, we will show that abstract models based on indiscriminability can completely replace the ground models for diagnostic reasoning without incurring any loss of relevant diagnostic information.

In the second part of the paper, we will describe in detail a practical algorithm for computing such a kind of abstractions, focusing on two specially important cases of indiscriminability (local and global-indiscriminability). Then, we will present a set of experimental results which confirm the expected benefits of adopting the computed abstractions for diagnosis, both in terms of the number of returned diagnoses and of computational cost. Finally, we will discuss the related work and conclude the paper with a discussion.

2. Problem characterization and state of the art

One of the main problems in using abstraction for diagnostic reasoning concerns the relation between ground and abstract diagnoses.

Figure 1 provides a graphical interpretation of the problem. The lower part of the figure shows the usual way of performing diagnosis, whereby the diagnostic problem is solved directly at the ground level by exploiting the ground model and obtaining the set of ground diagnoses and/or the preferred diagnoses according to criteria such as minimum cardinality or maximum probability.

The upper part shows that an abstract model is obtained by applying an *abstraction mapping* $\mathcal{AM}$ to the ground model. Also the observations may in general be abstracted by applying an abstraction mapping $\mathcal{AM}_{obs}$ (see, e.g., the variables domains abstractions in [22] for an example of abstract observations that differ from the ground ones). In this paper we aim at exploiting as much as possible the available observations in or-

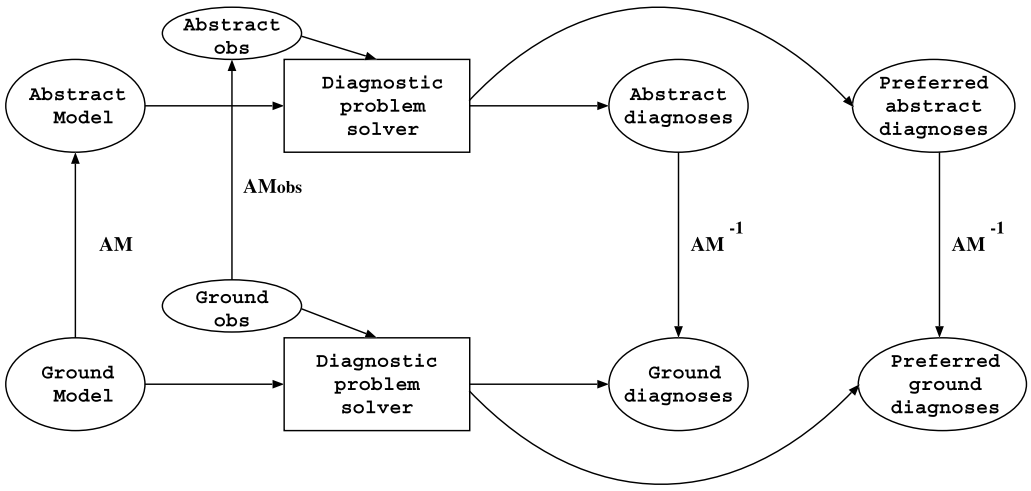

Fig. 1. Correspondence between ground and abstract diagnoses.

der not to loose diagnostic discrimination, and therefore we assume that abstract and ground observations are equal.

By exploiting the abstract model, it is possible to solve the diagnostic problem and directly obtain the set of abstract diagnoses and preferred diagnoses. The relation between abstract and ground (preferred) diagnoses is indicated by the downwards arrows in the picture. The question to be addressed is: how does the set of ground diagnoses obtained from the set of abstract diagnoses (through the inverse $\mathcal{AM}^{-1}$ of the mapping) relate to the actual set of ground diagnoses computed at the ground level?

Some important relations have been formalized in the literature as properties of the abstraction mapping [32]. In particular, the *downward failure* property of an abstraction mapping guarantees that each ground diagnosis is mapped to an abstract diagnosis; in this case, by solving the diagnostic problem at the abstract level and then mapping back to the ground level, we will not lose any ground diagnosis (but possibly we will obtain spurious diagnoses). The downward failure property is a prerequisite for the completeness of abstract diagnostic reasoning, and most of the methods developed so far aim at satisfying this property.

Another property formalized in [32] is the *upward-failure*, which guarantees that, if an assignment of behavioral modes to the system components is inconsistent at the ground level (i.e. not a ground diagnosis), it is necessarily mapped to an assignment that is inconsistent at the abstract level (i.e. not an abstract diagnosis). If the upward-failure property holds, one can solve a diagnostic problem at the abstract level, and when she maps back the abstract diagnoses to the ground ones, no spurious ground diagnoses are generated (but some diagnoses may be missed).

Obviously, the most interesting case is when both the downward- and the upward-failure properties hold, so that solving a diagnostic problem at the abstract level and then mapping back the abstract diagnoses gives the same set of ground diagnoses as the ones obtained by solving the diagnostic problem directly at the ground level.

Example 2.1. In order to show that it is not trivial to satisfy both the downward- and the upward-failure property, let us consider the simple digital circuit depicted in Fig. 2 (left), consisting of two buffers $B1$, $B2$, a NAND gate *NA* and two NOT gates $N1$, $N2$. Let us suppose that each component can be in mode *ok* or *ab* (where *ab* is any *abnormal* behavior that differs from the correct behavior defined by the usual truth-tables for the gates).

In this very simple example, the abstract component NA^* depicted in Fig. 2 (right) aggregates a subsystem composed by NAND gate *NA* and buffers $B1$, $B2$. A quite natural way to define the *ok* and *ab* modes of the abstract component is the following:[1]

$$NA^*(ok) \leftrightarrow B1(ok) \wedge B2(ok) \wedge NA(ok),$$

$$NA^*(ab) \leftrightarrow B1(ab) \vee B2(ab) \vee NA(ab)$$

i.e., NA^* is *ok* iff all its subcomponents are *ok*. It is easy to see that this abstraction exhibits the downward-

[1] Used in [13] for Theory Diagnoses and, implicitly, in [23] for the method based on cones.

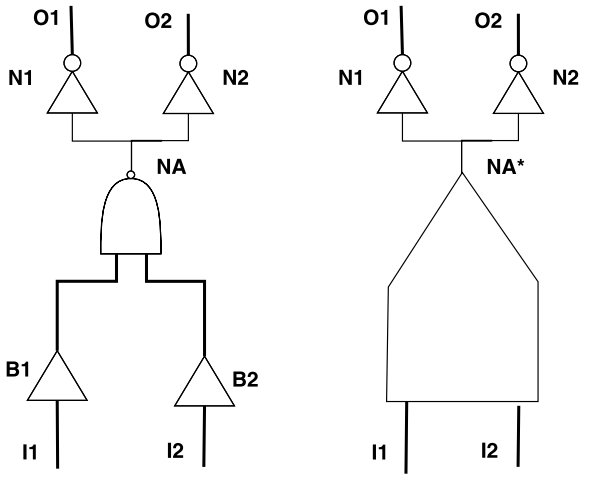

Fig. 2. A simple digital circuit with an abstraction.

failure property: if $\{B1(ok),\ B2(ok),\ NA(ok)\}$ is a ground diagnosis, it means that $B1(ok) \wedge B2(ok) \wedge NA(ok)$ is consistent with the observations (by definition of consistency-based diagnosis); therefore, $NA^*(ok)$ is consistent with the observations, i.e. it is an abstract diagnosis. Similarly, if any ground assignment that maps to $NA^*(ab)$ is a ground diagnosis, then $B1(ab) \vee B2(ab) \vee NA(ab)$ is consistent with the observations, and therefore $NA^*(ab)$ is an abstract diagnosis.

Unfortunately, however, the abstraction does not guarantee the upward-failure property. Indeed, it is easy to verify that, in a consistency based approach to diagnosis, the following abstract diagnosis explains the observations $\{I1(0), I2(0), O1(1), O2(1)\}$ on the system inputs and outputs:

$$\{NA^*(ab), N1(ok), N2(ok)\}.$$

However, the following ground assignment which maps to this diagnosis:

$$\{B1(ab), B2(ok), NA(ok), N1(ok), N2(ok)\}$$

is *not* a ground diagnosis, since given inputs $I1(0)$ and $I2(0)$ it produces outputs $O1(0)$, $O2(0)$. In other words, the abstraction introduces spurious diagnoses and therefore the upward-failure property does not hold.

The abstractions used in *hierarchical diagnosis* (e.g. [7], *SD-hierarchies* [19,23]) usually aim to guarantee the downward failure property in order to avoid missing some diagnoses, but do not guarantee the upward failure, since it is difficult to enforce both properties. Therefore, after computing the abstract diagnoses it is usually necessary to do further reasoning at the lower levels of the hierarchy in order to rule out spurious ground diagnoses; this process can be very time consuming, especially when the number of abstract diagnoses is very high and many of them turn out to generate only spurious diagnoses.

In recent years, Siddiqi and Huang [23] have shown that, for a limited class of system models that are directional and specify only the nominal behavior, it is possible to guarantee both the downward- and the upward-failure property for the set of preferred (minimum-cardinality) diagnoses. More specifically, their work addresses the diagnosis of digital circuits with a hierarchy of abstractions based on *cones*. A cone is a set of components dominated by a component called the cone *vertex*; all the paths from the subcomponents of the cone to the system outputs go necessarily through the vertex (for example, the abstract component NA^* in Fig. 2 is a cone dominated by the cone vertex NA).

The fundamental assumption made in the cone-based approach is that a single fault in the cone vertex can *always* explain the faulty behavior of the whole cone. Under this assumption, it can be proved that hierarchical reasoning will eventually compute exactly the set of preferred ground diagnoses. This is a relevant result for the diagnosis of digital circuits (and other directional systems with a weak fault model), and the cone-based approach has been shown to be able to efficiently diagnose very large circuits from the ISCAS-85 benchmark [5].

However, even for directional models such as those of digital circuits, the method proposed in [23] is guaranteed to work just in case the only behavior represented in the model is the correct behavior. In the following example we show the effect of introducing an explicit model of a faulty behavioral mode.

Example 2.2. Let us consider again the digital circuit represented in Fig. 2 and its abstraction. Let us also assume that each component has the *ok* mode and the faulty mode $sa1$ (stuck at one), which sets the output of the logical gate to 1 independently of the value of its inputs. The abstraction implicitly defined in [23] is:

$$NA^*(ok) \leftrightarrow B1(ok) \wedge B2(ok) \wedge NA(ok),$$
$$NA^*(ab) \leftrightarrow B1(sa1) \vee B2(sa1) \vee NA(sa1).$$

It is easy to see that the two preferred ground diagnoses of the diagnostic case characterized by the ob-

servations $\{I1(0), I2(0), O1(1), O2(1)\}$ are:

$$\{B1(sa1), B2(sa1), NA(ok), N1(ok), N2(ok)\},$$
$$\{B1(ok), B2(ok), NA(ok), N1(sa1), N2(sa1)\}.$$

Both of them involve a double fault. In contrast, if we solve the problem at the abstract level, the only preferred abstract diagnosis is:

$$\{NA^*(ab), N1(ok), N2(ok)\},$$

which involves only one fault. Note in particular that abstract diagnosis $\{NA^*(ok), N1(sa1), N2(sa1)\}$ is *not* preferred, since it involves two faults. The preferred ground assignments that map to the preferred abstract diagnosis are:

$$\{B1(sa1), B2(ok), NA(ok), N1(ok), N2(ok)\},$$
$$\{B1(ok), B2(sa1), NA(ok), N1(ok), N2(ok)\},$$
$$\{B1(ok), B2(ok), NA(sa1), N1(ok), N2(ok)\}.$$

Both of the actual preferred ground diagnoses are not among these assignments, and therefore are missed.

Our work is motivated by the goal of automatically performing component abstractions that exhibit both the downward- and upward-failure property on the whole set of diagnoses as well as on the minimum-cardinality diagnoses. More specifically, referring to the schema of Fig. 1, we aim to:

(1) compute a mapping $\mathcal{AM}$ that is applied off-line to the ground system model in order to obtain an abstract system model,
(2) use the abstract model on-line (each time we need to solve a diagnostic problem) for computing a set of abstract diagnoses that fully correspond to the ground diagnoses through $\mathcal{AM}^{-1}$.

As we shall see, abstractions are computed for a specific level of system observability and (possibly) system operating conditions. As far as such conditions do not change, the abstract model can be used for solving any diagnostic problem.[2] The advantage of computing and applying $\mathcal{AM}$ offline is twofold: first, by using a more abstract model, diagnostic problem solving is faster; second, we directly obtain abstract diagnoses that are fewer than the ground ones. It is worth noting, however, that an abstraction mapping $\mathcal{AM}$ can also be used in a different way. Referring again to the schema of Fig. 1, it is possible to use the ground model for computing the set of ground diagnoses, and then to apply $\mathcal{AM}$ to obtain the abstract diagnoses. This may be useful when applying $\mathcal{AM}$ to the system model leads to abstract components whose models are too complex, but we still want to compute abstract diagnoses because they are fewer and more informative.

Finally, it is worth stressing the fact that, compared to previous works on directional models such as [23, 28], we address the significantly broader class of finite-domain relational system models, where no directionality is required in the system behavior; a lot of attention has been devoted to this broader class in the Model-Based Diagnosis community [12,18,22,24], especially because it enhances reusability of component models [24]. Moreover, we handle strong fault models, that (possibly) specify the faulty behavior(s) of the components as well as the nominal one.

3. Preliminary definitions

Before presenting our definition of component abstractions, we precisely define the class of system models upon which the abstractions can be applied. As it is typical in MBD, we will first define component models and then use them to define system models.

Definition 3.1. A *Component Description* SD_c of component c is a pair $(\mathcal{SV}_c, DT_c)$ where:

- $\mathcal{SV}_c = \{c\} \cup P_c$ is a set of Finite-Domain (FD) variables associated with component c. The value of variable c represents the behavioral mode of the component (*ok* and one or more fault modes), while $P_c = \{p_1, \ldots, p_k\}$ represents a set of ports connecting c with other components or with the external world,
- DT_c (Component Domain Theory) is a relation on variables $\mathcal{SV}_c$ (i.e., $DT_c \subseteq dom(c) \times dom(p_1) \times \cdots \times dom(p_k)$) modeling the behavior of the component in the *ok* and faulty modes.

From the component descriptions, it is easy to derive a compositional definition of *System Description*.

Definition 3.2. Let $SD_1, \ldots, SD_n$ be a set of component descriptions. A *System Description SD* derived from $SD_1, \ldots, SD_n$ is a pair $(\mathcal{SV}, DT)$ where:

[2] Actually, it can be easily shown that an abstract model can be safely used also in case observability decreases.

- $\mathcal{SV} = (\bigcup_{i=1,\ldots,n} \mathcal{SV}_i)$ is the set of FD system variables partitioned in $C = \{c_1, \ldots, c_n\}$ (components), P (system ports) and I (internal variables). A set $O \subseteq P \cup I$ represents the set of observable variables (*observability*);
- $DT = \{DT_1, \ldots, DT_n\}$ (Domain Theory) is a set of component domain theories, each one modeling the behavior of a component.

The set of ports P represents the interface between the system and the external world, while the set of internal variables I represents the connections between pairs of components. Without loss of generality, we assume that each port $p \in P$ appears in exactly one component theory DT_i, while internal variables I appear in exactly two component theories DT_i, DT_j, $i \neq j$.

We denote as $P_{exo} \subseteq P \cap O$ the set of *exogenous* controllable ports, whose value is externally controlled and known, and with $P_{end} = P \backslash P_{exo}$ the remaining (endogenous) ports. An important role is played by system states, defined as follows.

Definition 3.3. Let *SD* be a System Description over components C. A *state* S of the system is an assignment $\{c_1(bm_1), \ldots, c_n(bm_n)\}$ of a behavioral mode to each component $c \in C$.

Given $DT = \{DT_1, \ldots, DT_n\}$, the *Global Domain Theory GDT* is defined as the natural join (denoted as $\bowtie$) of the component theories in *DT*:[3]

$$GDT = DT_1 \bowtie \cdots \bowtie DT_n.$$

The compositional approach adopted for building *SD* can be straightforwardly extended to model any subsystem Γ involving a subset of components $\{c_1, \ldots, c_m\} \subseteq C$. We will denote with $SD(\Gamma) = (\mathcal{SV}_\Gamma, DT_\Gamma)$ the model of subsystem Γ.

We are now ready to define *Diagnostic Problem*s and diagnoses.

Definition 3.4. A *Diagnostic Problem* is a pair $DP = (SD, \mathcal{O})$ where *SD* is a system description, and $\mathcal{O}$ is an assignment to the O variables.

[3]Since our modeling approach deals with relational models, all the operations needed for model manipulation will be expressed in terms of the standard operators of relational algebra: $\bowtie$, *project* and *select*.

Definition 3.5. Let $DP = (SD, \mathcal{O})$ be a Diagnostic Problem. A *consistency-based* diagnosis for DP is a system state S_D such that:

$$GDT \bowtie S_D \bowtie \mathcal{O} \neq \emptyset.$$

The definition reported above corresponds to the classical definition of consistency based diagnosis:

$$GDT \wedge S_D \wedge \mathcal{O} \not\vdash \bot$$

expressed in relational rather than logical terms: S_D is a diagnosis for a problem $DP = (SD, \mathcal{O})$ iff S_D and $\mathcal{O}$ appear together in at least one tuple of *GDT*, i.e. if $\mathcal{O}$ is a possible observation induced by S_D.

We also define a preference criterion among diagnoses, by enriching the model with ranks $r(c(bm))$ associated with each behavioral mode *bm* of each system component c. Ranks [15] are non-negative integers s.t., informally speaking, an event with rank r is one order-of-magnitude more likely than an event with rank $r + 1$; since smaller ranks represent higher probabilities, a rank can be seen as a cost.

Under the usual assumption of independence of faults, the rank of a diagnosis $S_D = \{c_1(bm_1), \ldots, c_n(bm_n)\}$ is given by:

$$r(S_D) = \sum_{i=1}^{n} r\big(c_i(bm_i)\big)$$

and we prefer the diagnoses with the smallest rank.

It is easy to see that, when the *ok* mode of each component c has rank 0 and all the other (faulty) modes have rank 1, our preference criterion corresponds to minimum cardinality. For simplicity, we will assume that ranks are assigned in this way in the ground model, so that a ground diagnosis of cardinality N has rank N. In general, however, the rank-based criterion is more flexible than minimum cardinality, since it allows different fault modes of components to have different weights. This possibility will be necessary for properly defining the ranks of abstract behavioral modes.

4. Abstractions

4.1. Abstraction mappings

We are interested in abstracting subsystems into abstract components. Since, as a result of abstraction, the internal variables of the subsystem are dis-

carded (as explained in Section 4.2), we will consider only subsystems whose internal variables are all non-observable (i.e. such that $O \cap I(\Gamma) = \emptyset$); in this way, the observations are preserved by the abstraction.

In our framework, components have in general many behavioral modes (Definition 3.1) and, therefore, the abstraction must specify not only which components Γ are merged into an abstract component AC_Γ, but also how the behaviors of the components of subsystem Γ are abstracted into the behavior of AC_Γ. The following definition of *Component Abstraction Mapping* takes care of these important points.

Definition 4.1. Let *SD* be a system description with observability O, and Γ be a subsystem s.t. $O \cap I(\Gamma) = \emptyset$. A *Component Abstraction Mapping* $\mathcal{AM}_\Gamma$ is a triple $(AC_\Gamma, \Gamma, BM_\Gamma)$ where:

- AC_Γ is the abstract component defined by the mapping, with ports $P(AC_\Gamma) = P(\Gamma)$,
- $\Gamma = \{c_1, \ldots, c_m\}$ is the set of *subcomponents* of AC_Γ,
- BM_Γ is a set $\{(abm_1, \lambda_1), \ldots, (abm_k, \lambda_k)\}$, which:
  - enumerates the behavioral modes $abm_1, \ldots, abm_k$ of AC_Γ,
  - associates with each abm_i a non-empty subset λ_i of states of Γ s.t. any possible state S_Γ of Γ is associated with exactly one abstract behavioral mode abm_i.

The ports of the abstract component AC_Γ are exactly the same as the ones of the subsystem Γ, while the internal variables of Γ are forgotten in the abstract component AC_Γ (because abstract components, as ground components, do not have internal variables). The definition imposes the obvious requirement that each abm_i corresponds to at least one state of Γ. More interestingly, it also requires that the abstract behavioral modes cover all the ground states of Γ and are mutually exclusive, as it is the case for ground behavioral modes.

Component abstraction mappings, by definition, map each state of a subsystem to some abstract behavioral mode. As we shall see, it is also useful to introduce mappings that are defined just on the preferred states according to the definitions given in Section 3.

Definition 4.2. Let $\mathcal{AM}_\Gamma$ be a Component Abstraction Mapping defining an abstract component AC_Γ with behavioral modes $\{abm_1, \ldots, abm_k\}$. The rank of abstract behavioral mode abm_i associated with the set of states λ_i is defined as:

$$r(abm_i) = \min\big(\{r(S_\Gamma)\colon\ S_\Gamma \in \lambda_i\}\big).$$

A *Preferred Component Abstraction Mapping* $\mathcal{PM}_\Gamma$ derived from $\mathcal{AM}_\Gamma$ associates with each $abm_i \in dom(AC_\Gamma)$ the set of states $\lambda'_i = \{S_\Gamma\colon\ S_\Gamma \in \lambda_i$ and $r(S_\Gamma) = r(abm_i)\}$.

According to this definition, a preferred component abstraction mapping is just a restriction of a component abstraction mapping to the preferred states of Γ. Note that the definition of the rank of an abstract behavioral mode abm_i follows from the fact that abm_i represents the disjunction of all the states $S_\Gamma \in \lambda_i$, whose rank is the *min* of the ranks of such states.

The notion of abstraction mapping can be straightforwardly extended to the whole system, by considering a set of component abstraction mappings.

Definition 4.3. Let us consider a set of subsystems $\Gamma^1, \ldots, \Gamma^p$ such that they form a partition of the whole system (more formally, $\Gamma^1 = \{c^1_1, \ldots, c^1_{m_1}\}, \ldots, \Gamma^p = \{c^p_1, \ldots, c^p_{m_p}\}$ are a partition of the set of components C).

A (System) *Abstraction Mapping* $\mathcal{AM}$ is a set of Component Abstraction Mappings $\{\mathcal{AM}_{\Gamma^1}, \ldots, \mathcal{AM}_{\Gamma^p}\}$ which maps each subsystem Γ^i to an abstract component AC_{Γ^i} $(i = 1, \ldots, p)$.

We conclude this section by defining a partial order among component abstraction mappings that will prove useful later in the paper.

Definition 4.4. Let us consider two Component Abstraction Mappings $\mathcal{AM}_\Gamma$ and $\mathcal{AM}'_\Gamma$ of a subsystem Γ such that $\mathcal{AM}_\Gamma$ partitions the set of states of Γ in $\{\lambda_1, \ldots, \lambda_k\}$ and $\mathcal{AM}'_\Gamma$ in $\{\lambda'_1, \ldots, \lambda'_l\}$. If for each λ'_i, $i \in \{1, \ldots, l\}$ there exists λ_j, $j \in \{1, \ldots, k\}$ s.t. $\lambda'_i \subseteq \lambda_j$, we say that $\mathcal{AM}'_\Gamma$ is a refinement of $\mathcal{AM}_\Gamma$.

Intuitively, $\mathcal{AM}_\Gamma$ represents a stronger abstraction of subsystem Γ, since each of the abstract behavioral modes it defines potentially corresponds to two or more abstract behavioral modes defined by its refinement $\mathcal{AM}'_\Gamma$.

4.2. Abstract system descriptions

Starting from a component abstraction mapping $\mathcal{AM}_\Gamma = (AC_\Gamma, \Gamma, BM_\Gamma)$, it is possible to automatically synthesize the component theory $DT(AC_\Gamma)$ which describes the behavior of the abstract component AC_Γ. Let $GDT_P(\Gamma)$ be the relation obtained by discarding the internal variables from the global domain theory of Γ. In relational algebra, this corresponds to projecting $GDT(\Gamma)$ on the components and ports of Γ, i.e.:

$$GDT_P(\Gamma) = \mathbf{project}_{\Gamma \cup P(\Gamma)} GDT(\Gamma).$$

Each tuple of $GDT_P(\Gamma)$ has the form $S_\Gamma \cup \mathcal{P}_\Gamma$, where:

- S_Γ is a state of Γ (i.e. an assignment of behavioral modes to the components of the subsystem),
- $\mathcal{P}_\Gamma$ is an assignment to the ports $P(\Gamma)$ of Γ.

In order to obtain $DT(AC_\Gamma)$, we simply replace each tuple $S_\Gamma \cup \mathcal{P}_\Gamma$ in $GDT_P(\Gamma)$ with a tuple $\{AC_\Gamma(abm)\} \cup \mathcal{P}_\Gamma$ such that $\mathcal{AM}_\Gamma$ associates state S_Γ with the abstract behavioral mode *abm*. Note that the resulting component theory of AC_Γ is expressed in terms of the variable AC_Γ representing the abstract component, plus the ports of the subsystem Γ.

It is important to note that:

- the resulting component theory $DT(AC_\Gamma)$ of AC_Γ is not larger than the global domain theory $GDT(\Gamma)$ of the abstracted subsystem Γ. Indeed, both the projection on the $\Gamma \cup P$ variables and the replacement of states S_Γ with $AC_\Gamma(abm)$ can transform sets of tuples into single tuples. In practice, this can cause a dramatic reduction of the number of tuples (see Table 1 and the associated discussion in Section 8),
- the number of states $dom(AC_\Gamma)$ of component AC_Γ is not larger than the number of states of the abstracted subsystem Γ. Indeed, there is at most one behavioral mode of AC_Γ for each state of Γ.

Note that, as a consequence, the number of tuples in the system global domain theory GDT does not increase (and, in practice, can significantly decrease) by applying abstractions. Indeed, GDT can be written as:

$$GDT = GDT(\Gamma) \bowtie GDT(C \backslash \Gamma)$$

i.e., as the join of the global domain theories of subsystem Γ and its complementary subsystem $C \backslash \Gamma$. If $GDT(\Gamma)$ is replaced with $DT(AC_\Gamma)$, GDT does not increase. Similarly, the number of global system states can be written as:

$$\mathcal{S} = \mathcal{S}(\Gamma) \times \mathcal{S}(C \backslash \Gamma)$$

i.e., as the cross product of the sets of states of Γ and $C \backslash \Gamma$. If $\mathcal{S}(\Gamma)$ is replaced with $dom(AC_\Gamma)$, $\mathcal{S}$ does not increase. Since $\mathcal{S}$ represents the search space for diagnostic reasoning, also the number of solutions to diagnostic problems cannot be negatively affected by abstraction. Note that, in general, the size of $\mathcal{S}$ depends on the number of system components and on the number of behavioral modes of each component; since component abstraction addresses both of these parameters, it can cause a dramatic reduction of the diagnostic search space (see Table 2 and the associated discussion in Section 8).

What abstraction does not guarantee *per se* is that the number of tuples in $DT(AC_\Gamma)$ is always small enough to be stored and managed efficiently; for this reason, in Section 7 we will add further conditions for accepting and applying a candidate component abstraction; when such conditions are not met, the abstraction is simply discarded, and other candidate abstractions are considered.

With the component theories of abstract components, it is easy to build an abstract system description $SD_A = (\mathcal{SV}_A, DT_A)$ corresponding to a given ground system description $SD = (\mathcal{SV}, DT)$ and a system abstraction mapping $\mathcal{AM} = \{\mathcal{AM}_{\Gamma^1}, \ldots, \mathcal{AM}_{\Gamma^p}\}$.

The set of abstract system variables $\mathcal{SV}_A$ is partitioned in P_A (ports), C_A (components) and I_A (internal variables) where:

- $C_A = \{AC_{\Gamma^1}, \ldots, AC_{\Gamma^p}\}$,
- $P_A = P$,
- $I_A = (\bigcup_{i=1,\ldots,p} P(\Gamma^i)) \backslash P_A$.

First of all, the components in SD_A are the abstract components $AC_{\Gamma^1}, \ldots, AC_{\Gamma^p}$ specified in the abstraction mapping. The ports of the abstract system are the same as the ones of the ground system, while the internal variables I_A are the union of the sets of ports of the abstract components $AC_{\Gamma^1}, \ldots, AC_{\Gamma^p}$, except for the variables which are system ports.

Finally, the abstract domain theory DT_A is just the set of component theories modeling the behavior of each abstract component, i.e. $DT_A = \{DT(AC_{\Gamma^1}), \ldots, DT(AC_{\Gamma^p})\}$.

Example 4.1. To illustrate the concepts introduced so far, let us now focus our attention on a more signif-

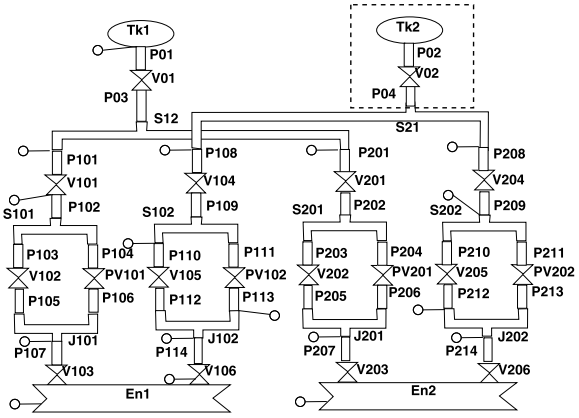

Fig. 3. The schema of the propulsion subsystem and the HIGH observability.

icant system. In Fig. 3 we report the overall schema of the propulsion system of a spacecraft adapted from the one discussed in [18] to illustrate NASA's Model-Based approach to spacecraft autonomy developed for the Remote Agent experiment.

The system involves 64 components: two tanks ($Tk1$ and $Tk2$), two engines ($En1$ and $En2$), 14 valves ($V01$, $V02$, $V101$–$V106$, $V201$–$V206$), 4 pyro-valves ($PV101$, $PV102$, $PV201$, $PV202$), 6 splits ($S12$, $S21$, $S101$, $S102$, $S201$, $S202$), 4 joins ($J101$, $J102$, $J201$, $J202$) and 32 pipes. As it is typically done in Model Based Diagnosis, the component models are specified by instantiating the generic models of certain types of components. Appendix A provides a detailed description of the component models involved in the propulsion system, which follow the modeling approach described in [24]. Here we just stress the fact that the component models are non-directional (i.e. there is no distinction between inputs and outputs of a component).

As for the fault modes of the components, tanks and junctions are assumed to have no faults; pipes can be clogged, denoted *cl* (flow is completely blocked by the pipe) or broken, denoted *br* (flow that enters one end of the pipe does not reach the other end); and valves can be stuck-open, denoted *so* (the valve is open even if it has been commanded to close) and stuck-closed, denoted *sc* (the valve is closed even if it has been commanded to open).

The propulsion system is also characterized by 18 exogenous commands, namely the commands to the 14 valves and to the four pyro-valves:

$$P_{exo} = \{cmd_{V01}, \ldots, cmd_{V206}, cmd_{PV101}, \ldots, cmd_{PV202}\}.$$

In principle the (potentially) observable endogenous variables include the flow at each pipe terminal, the flows at each engine terminal and the thrust of the two engines (for a total of 70 potential observables).

The system observability, denoted as *HIGH* for later reference and partially indicated in the picture by white circles, includes:

- all of the commands cmd_V issued to the valves and pyro-valves, which can take values *open* or *close*
- endogenous variables $\{f_1^{P01}, f_1^{P101}, f_1^{P102}, f_1^{P107}, f_1^{P108}, f_1^{P110}, f_2^{P113}, f_1^{P114}, f_2^{En1}, f_1^{P201}, f_1^{P207}, f_1^{P208}, f_2^{P209}, f_2^{P212}, f_1^{P214}, th_1, th_2\}$, where:
  - each variable f_i^c represents flow at terminal i of component c, and can take values 0 (no flow), $+$ (flow enters c through terminal i) and $-$ (flow leaves c through terminal i),
  - variables th_1, th_s represent the presence of thrust at each engine, and can take values *yes* and *no*.

Overall, the system description of the ground model of the propulsion system involves 494 variables and it is easy to imagine that the diagnosis of such a system may produce a large number of results (even of preferred diagnoses) which may be difficult to understand by the human (or artificial) operator in charge of monitoring the system.

In the top right portion of Fig. 3, a dashed box encloses a subsystem Γ composed by tank $Tk2$, pipe $P02$, valve $V02$, and pipe $P04$. Let us consider the abstraction of this subsystem into an abstract component $Tk2 \rightarrow P04$. According to Definition 4.1, the ports of $Tk2 \rightarrow P04$ include the variables associated with the second end of pipe $P04$ (in particular, the flow f_2^{P04} exiting from $P04$), while all the variables associated with the terminals connecting the other components are discarded during abstraction; moreover, the abstract component has a port corresponding to the exogenous command to valve $V02$.

This abstraction is allowed by our definition of component abstraction mapping, since none of the internal variables of subsystem $\{Tk2, P02, V02, P04\}$ are observable. This is not true for the corresponding subsystem involving $Tk1$, $P01$, $V01$ and $P03$ (top left portion of Fig. 3), since the terminal connecting $Tk1$ and $P01$ is observable.

It is important to stress the fact that, in our framework, the structure of the abstraction (i.e. the set of

$\mathcal{AM}$:
$abm0 = \{(ok, ok, ok, ok)\}$
$abm1 = \{(ok, ok, so, ok)\}$
$abm2 = \{(ok, ok, ok, cl), (ok, ok, ok, br), (ok, ok, so, cl), (ok, ok, so, br), (ok, ok, sc, ok),$
$(ok, cl, ok, ok), (ok, br, ok, ok), (ok, cl, so, ok), (ok, br, so, ok), (ok, cl, sc, ok), (ok, br.sc, ok),$
$(ok, ok, sc, cl), (ok, cl, ok, cl), (ok, br, ok, cl), (ok, cl, so, cl), (ok, br, so, cl), (ok, cl, sc, cl),$
$(ok, br, sc, cl), (ok, ok, sc, br), (ok, cl, ok, br), (ok, br, ok, br), (ok, cl, so, br), (ok, br, so, br),$
$(ok, cl, sc, br), (ok, br, sc, br)\}$

$\mathcal{PM}$:
$abm0' = \{(ok, ok, ok, ok)\}$
$abm1' = \{(ok, ok, so, ok)\}$
$abm2' = \{(ok, ok, ok, cl), (ok, ok, ok, br), (ok, ok, sc, ok), (ok, cl, ok, ok), (ok, br, ok, ok)\}$

Fig. 4. An abstraction mapping and its Preferred Mapping. A tuple (x, y, w, z) stands for a state $\{Tk2(x), P02(y), V02(w), P04(z)\}$ of the abstracted subsystem.

components Γ that have been merged into an abstract component AC_Γ) is not sufficient to completely capture the mapping. According to Definition 4.1, we also need to specify the abstract behavioral modes abm_i of AC_Γ and their associated sets of states λ_i.

Each one of the 27 states of subsystem $\Gamma = \{Tk2, P02, V02, P04\}$ (recall that pipes can be in modes *ok*, *cl*, *br*, and valves can be in modes *ok*, *so*, *sc*, while tanks are assumed to be always in the *ok* mode) must be mapped to exactly one abstract behavioral mode of $Tk2 \rightarrow P04$.

As we will show in the next sections, an interesting abstraction mapping $\mathcal{AM}$ is the one shown in Fig. 4 (top), which involves just three abstract behavioral modes. It is easy to verify that this abstraction mapping represents a partition of the set of possible states of subsystem $\Gamma = \{Tk2, P02, V02, P04\}$. The abstract behavioral mode *abm*0 corresponds to the case where all components are *ok*, *abm*1 corresponds to the case where there is some flow exiting from $P04$ even if the command to $V02$ is set to close, and *abm*2 corresponds to the many situations where there is no flow from $P04$.

We have also introduced the notion of preferred abstraction mapping (Definition 4.2), where only the preferred states of Γ are retained. For the abstract component $Tk2 \rightarrow P04$, the preferred abstraction mapping $\mathcal{PM}$ corresponding to the abstraction mapping discussed above is shown in Fig. 4 (bottom).

The rank of *abm*0 is 0, while the rank of *abm*1 and *abm*2 is 1. Note in particular that in the mapping $\mathcal{PM}$, the only ground states mapped to the abstract mode *abm*2 are those of rank 1.

5. Indiscriminability

5.1. Global indiscriminability

In many cases, the solution to a diagnostic problem *DP* involves a large set of diagnoses. Obviously, the number of alternative diagnoses strongly depends on the observations. In fact, when in a specific diagnostic problem *DP* the observations $\mathcal{O}$ involve just a small subset of the variables, we can expect a large set of diagnoses due to the limited ability to discriminate among different diagnoses.

A small discrimination power has a (negative) impact not only on the time needed for computing the large set of diagnoses, but also on the information value of the result for the human or artificial supervisor responsible for taking an action (e.g., interpretation, reconfiguration, repair) once the diagnosis has been performed.

In this section we introduce *indiscriminability* to formally capture the notion of discrimination power. We relate indiscriminability with two factors that can have a strong impact: the system observability O (as discussed above), and the *context restriction* $\mathcal{R}$. Contexts and context restrictions are defined as follows.

Definition 5.1. A *context* is an assignment $\mathcal{X}$ to the P_{exo} variables.

Recalling that the P_{exo} variables represent the exogenous controllable ports, such as the inputs to a circuit or the commands to a set of valves, a context is simply a specific choice of values for such controllable ports, e.g., a specific input vector to a circuit or a specific set of commands to the valves.

Definition 5.2. A *context restriction* $\mathcal{R}$ specifies the assignment of a value to each variable in a subset R of P_{exo}. The meaning of a context restriction is that the system will operate only in contexts $\mathcal{X}$ that agree with the partial assignment $\mathcal{R}$.

In practice, context restrictions are useful mainly to specify restrictions on the exogenous commands that can be issued to the system.

We start with the definition of a special case of indiscriminability, *G-indiscriminability*, that will prove to be particularly important in the following. Then, in the next section we generalize it to the broader case of Γ^+-indiscriminability.

Definition 5.3. Let *SD* be a system description, O be the system observability and $\mathcal{R}$ be a context restriction. Moreover, let Γ be a subsystem s.t. $I(\Gamma) \cap O = \emptyset$.

We say that two states S_Γ, S'_Γ of Γ are *G-indiscriminable* w.r.t. O, $\mathcal{R}$ iff for any state S_Δ of $\Delta = C \backslash \Gamma$ the following holds:

$$\begin{aligned}&\mathbf{project}_O(\mathbf{select}_{S_\Gamma \wedge \mathcal{R} \wedge S_\Delta} GDT)\\ &\quad = \mathbf{project}_O(\mathbf{select}_{S'_\Gamma \wedge \mathcal{R} \wedge S_\Delta} GDT).\end{aligned} \tag{1}$$

In Eq. (1) we check the equality of two relations ρ and ρ', obtained by selecting from the global system model *GDT* the tuples that agree with assignments S_Γ, $\mathcal{R}$ and S_Δ (resp. S'_Γ, $\mathcal{R}$ and S_Δ for ρ'), and then projecting such tuples on the system observables O. In other words, the G-indiscriminability of S_Γ, S'_Γ requires that, in the contexts restricted by $\mathcal{R}$, these two states produce the same system observations regardless of the state of the other system components $\Delta = C \backslash \Gamma$.

There is an obvious relation between the solutions to a diagnostic problem and the notion of G-indiscriminability. Let us consider a diagnostic problem $DP = (SD, \mathcal{O})$ where $\mathcal{O}$ respects the constraints imposed by a context restriction $\mathcal{R}$. If states S_Γ, S'_Γ of Γ are G-indiscriminable w.r.t. O, $\mathcal{R}$ and if $D = S_\Gamma \cup S_\Delta$ is a diagnosis for the diagnostic problem *DP*, then $D' = S'_\Gamma \cup S_\Delta$ is an alternative diagnosis for *DP*. More generally, if the system has been partitioned into subsystems $\Gamma^1, \ldots, \Gamma^p$, and $D = S_{\Gamma^1} \cup \cdots \cup S_{\Gamma^p}$ is a diagnosis for a diagnostic problem *DP*, all of the diagnoses $D' = S'_{\Gamma^1} \cup \cdots \cup S'_{\Gamma^p}$ s.t. S_{Γ^i}, S'_{Γ^i} $(i = 1, \ldots, p)$ are G-indiscriminable w.r.t. O, $\mathcal{R}$ are also diagnoses for *DP*. Depending on how many states are G-indiscriminable from S_{Γ^i} in each subsystem Γ^i, the set of diagnoses that are *always* computed and returned with diagnosis D can be very large (in fact, up to exponential in the number of components $|C|$).

5.2. Generalization of G-indiscriminability

Before generalizing G-indiscriminability, we need to introduce the concept of *obs-boundary* of a subsystem Γ.

Definition 5.4. Let *SD* be a system description, O be the system observability and Γ be a subsystem. The *obs-boundary* of Γ is defined as:

$$B(\Gamma) = O(\Gamma) \cup P(\Gamma).$$

Intuitively, the obs-boundary of Γ contains all of the variables associated with subsystem Γ that could impact the diagnosis of Γ itself: the observables within Γ, and the ports that connect Γ with the rest of the system (whose values can, therefore, be related with the values of some observables outside of Γ). The obs-boundary $B(\Gamma)$ plays for subsystem Γ the same role that the system observability O plays for the whole system.

Definition 5.5. Let *SD* be a system description, O be the system observability and $\mathcal{R}$ be a context restriction. Moreover, let Γ be a subsystem s.t. $I(\Gamma) \cap O = \emptyset$, and let Γ^+ be a subsystem containing Γ, i.e. $\Gamma \subseteq \Gamma^+$.

We say that two states S_Γ, S'_Γ of Γ are Γ^+-*indiscriminable* w.r.t. O, $\mathcal{R}$ iff for any state S_{Δ^+} of $\Delta^+ = \Gamma^+ \backslash \Gamma$ the following holds:

$$\begin{aligned}&\mathbf{project}_{B(\Gamma^+)}\big(\mathbf{select}_{S_\Gamma \wedge \mathcal{R} \wedge S_{\Delta^+}} GDT(\Gamma^+)\big)\\ &\quad = \mathbf{project}_{B(\Gamma^+)}\big(\mathbf{select}_{S'_\Gamma \wedge \mathcal{R} \wedge S_{\Delta^+}} GDT(\Gamma^+)\big).\end{aligned} \tag{2}$$

This definition mirrors Definition 5.3 of G-indiscriminability; however, the considered portion of the system surrounding Γ is generalized from being the whole set of components C to any subsystem $\Gamma^+ \supseteq \Gamma$; as a consequence, the global system model *GDT* is replaced by the global model $GDT(\Gamma^+)$ of subsystem Γ^+, and the system observability O is replaced to the obs-boundary $B(\Gamma^+)$. Note that the Γ^+-indiscriminability relations (including G-indiscriminability) are equivalence relations, and therefore they induce a partition of the set of possible states of subsystem Γ into indiscriminability classes.

The generalization of G-indiscriminability to Γ^+-indiscriminability determines a spectrum of indiscriminability relations, one for each admissible Γ^+. In the following, beside G-indiscriminability, we will focus on another special case of Γ^+-indiscriminability, namely *L-indiscriminability* (local indiscriminability).

L-indiscriminability is characterized by the fact that $\Gamma^+ = \Gamma$, i.e. only the subsystem Γ is taken into account while the rest of the system is ignored. In this case, $B(\Gamma^+) = B(\Gamma) = P(\Gamma)$ (recall that $I(\Gamma) \cap O = \emptyset$), and therefore the projections are made on the ports of Γ. In other words, the L-indiscriminability of S_Γ, S'_Γ requires that (in the contexts restricted by $\mathcal{R}$) these two states produce the same values on the ports of Γ.

The notions of G- and L-indiscriminability as defined here are strongly related to the notions of global/local interchangeability exploited in the compilation of CSPs [31] and also in MBD [22]. We prefer to use the different term *indiscriminability* to stress the fact that indiscriminability applies to states S_Γ, S'_Γ of a subsystem Γ instead of tuples of some relation, and it depends on the system observability and the context restriction; these differences are important in order to better suit the characteristics of the diagnostic task. Despite these differences, we believe that in future work it could be worth exploring the adaptation of techniques developed for determining interchangeability to our framework (e.g., [25,31]). In particular, adapting the *tree-of-BDDs* approach described in [25] could likely be useful for improving the scalability of the BDD-based computation of G-indiscriminability that will be described in Section 7.5.

We conclude this section by defining a convenient notation for denoting two special kinds of component abstraction mappings, based on G- and L-indiscriminability, that will be important in the following.

Definition 5.6. Let $(AC_\Gamma, \Gamma, BM_\Gamma)$ be a component abstraction mapping with $BM_\Gamma = \{(abm_1, \lambda_1), \ldots, (abm_k, \lambda_k)\}$.

If the sets λ_i associated with each abm_i correspond one-to-one with the G-indiscriminability classes of the states of Γ, i.e.:

$$S_\Gamma, S'_\Gamma \in \lambda_i \quad \text{iff } S_\Gamma, S'_\Gamma \text{ are G-indiscriminable}$$

then we denote such a mapping as $\mathcal{AM}^G_\Gamma$.

Similarly, we denote as $\mathcal{AM}^L_\Gamma$ a component abstraction mapping corresponding one-to-one with the L-indiscriminability classes.

5.3. Correct abstraction mappings

Based on Γ^+-indiscriminability, we now specialize the notion of component abstraction mapping (Definition 4.1) to that of *correct* component abstraction mapping. We say that such mappings are correct because they guarantee that diagnostic reasoning at the abstract level does not introduce spurious (i.e. incorrect) ground diagnoses, as shown in Section 6.

Definition 5.7. A Component Abstraction Mapping $\mathcal{AM}_\Gamma$ is *correct* w.r.t. the degree of observability O and a context restriction $\mathcal{R}$ iff any pair of states of Γ that are associated by $\mathcal{AM}_\Gamma$ with the same abstract behavioral mode abm_i are Γ^+-indiscriminable for some $\Gamma^+ \supseteq \Gamma$.

From the definition it is immediate to see that $\mathcal{AM}^G_\Gamma$ and $\mathcal{AM}^L_\Gamma$ are correct, as well as any mapping which associates the abstract behavioral modes with the Γ^+-indiscriminability classes one-to-one for some $\Gamma^+ \supseteq \Gamma$. The following theorem assures that $\mathcal{AM}^G_\Gamma$ is the strongest among the correct abstraction mappings. The proof is reported in Appendix C.

Theorem 5.1. *Each correct abstraction mapping is a refinement of* $\mathcal{AM}^G_\Gamma$.

Example 5.1. Consider again the hydraulic system of Figure 3, and in particular the subsystem $\Gamma = \{Tk2, P02, V02, P04\}$.

If we apply the G-indiscriminability relation, we get an abstraction mapping $\mathcal{AM}^G_\Gamma$ which is exactly the one reported in the previous Example 4.1; such a mapping defines just three abstract behavioral modes where:

$$abm0^G = \{(ok, ok, ok, ok)\},$$
$$abm1^G = \{(ok, ok, so, ok)\},$$
$$abm2^G \text{ includes the other ground states of } \Gamma.$$

If, on the other hand, we make use of the notion of local indiscriminability, we get the abstraction mapping $\mathcal{AM}^L_\Gamma$ shown in Fig. 5, which involves 5 abstract behavioral modes.

It is easy to see that $\mathcal{AM}^L_\Gamma$ is indeed a refinement of $\mathcal{AM}^G_\Gamma$ and, in particular, that modes $abm2^L$, $abm3^L$, and $abm4^L$ refine mode $abm2^G$.

6. Correspondence of diagnoses

We are now in the position of addressing some of the issues discussed in Section 2, and graphically illustrated in Fig. 1. In particular, we start from the question whether, by solving the diagnostic problem at the abstract level and then mapping back the abstract diagnoses with the inverse mapping $\mathcal{AM}^{-1}$, we can get the

$$
\begin{aligned}
abm0^L &= \{(ok, ok, ok, ok)\} \\
abm1^L &= \{(ok, ok, so, ok)\} \\
abm2^L &= \{(ok, br, ok, ok)\} \\
abm3^L &= \{(ok, ok, ok, cl), (ok, ok, so, cl), (ok, ok, sc, ok), (ok, cl, ok, ok), (ok, cl, so, ok), \\
&\quad (ok, cl, sc, ok), (ok, br, sc, ok), (ok, ok, sc, cl), (ok, cl, ok, cl), (ok, cl, so, cl), (ok, cl, sc, cl), \\
&\quad (ok, br, sc, cl), (ok, br, so, cl), (ok, br, ok, cl)\} \\
abm4^L &= \{(ok, ok, ok, br), (ok, ok, so, br), (ok, ok, sc, br), (ok, cl, ok, br), (ok, cl, so, br), \\
&\quad (ok, cl, sc, br), (ok, br, ok, br), (ok, br, so, ok), (ok, br, so, br), (ok, br, sc, br)\}
\end{aligned}
$$

Fig. 5. An abstraction mapping based on local indiscriminability. Each tuple (x, y, w, z) stands for a state $\{Tk2(x), P02(y), V02(w), P04(z)\}$.

same set of ground diagnoses as if we solved the diagnostic problem directly at the ground level. After answering (positively) this question, we will answer the same question for preferred diagnoses.

The possibility of performing diagnostic reasoning at the abstract level without losing any ground diagnosis is assured by the following theorem.

Theorem 6.1. *Let $\mathcal{AM}$ be a System Abstraction Mapping according to Definition* 4.3. *Moreover, let SD_A be the abstraction of SD according to $\mathcal{AM}$ and let $DP = (SD, \mathcal{O})$ be a diagnostic problem; we denote with $DP_A = (SD_A, \mathcal{O})$ the abstract diagnostic problem corresponding to DP according to $\mathcal{AM}$.*

For any ground state S_G that is a diagnosis for DP, the abstract state S_A obtained from S_G via $\mathcal{AM}$ is an abstract diagnosis for DP_A.

The proof is reported in Appendix C. The theorem states that each diagnosis obtained by solving a ground diagnostic problem *DP* corresponds to an abstract diagnosis obtained by solving a diagnostic problem DP_A at the abstract level, so we are guaranteed that the *downward-failure* property holds.

It is worth noting that the precondition of this theorem is quite weak: in fact, according to Definitions 4.1 and 4.3, a system abstraction mapping requires only that the abstract modes of an abstract component are associated with a partition of the ground states of the abstracted subsystem. This explains why guaranteeing the *downward-failure* property alone is relatively easy, so that most of the existing approaches to hierarchical abstraction are actually able to enforce it.

While Theorem 6.1 focused on a condition for not missing diagnoses, in the following theorem we show that, if the abstraction mapping is correct, no spurious ground diagnoses are introduced by reasoning at the abstract level.

Theorem 6.2. *Let $\mathcal{AM}$ be a correct System Abstraction Mapping according to Definition* 5.7. *Moreover, let SD_A be the abstraction of SD according to $\mathcal{AM}$ and let $DP = (SD, \mathcal{O})$ be a diagnostic problem; we denote with $DP_A = (SD_A, \mathcal{O})$ the abstract diagnostic problem corresponding to DP according to $\mathcal{AM}$.*

For any abstract state S_A that is an abstract diagnosis for DP_A, all of the ground states S_G that can be obtained from S_A via the inverse abstraction mapping $\mathcal{AM}^{-1}$ are diagnoses for DP.

The proof is reported in Appendix C. This theorem guarantees that the *upward-failure* property holds if $\mathcal{AM}$ is correct w.r.t. O and $\mathcal{R}$. It is worth noting that the precondition of this theorem is significantly more demanding than that of the previous theorem, since it requires that the abstraction mapping is *correct*, and therefore it involves the satisfaction of the indiscriminability relation (see Definition 5.7).

Together, Theorems 6.1 and 6.2 guarantee that, for correct abstraction mappings, we can answer positively to the main question posed by the schema of Fig. 1: solving a diagnostic problem at the abstract level and then mapping back the abstract diagnoses gives the same set of ground diagnoses obtained by solving the diagnostic problem directly at the ground level.

Actually, we are able to prove a much stronger result: considering a correct abstraction mapping w.r.t. observability O and context restriction $\mathcal{R}$, is not only a sufficient condition for Theorem 6.2 to hold, but it is also a necessary condition, as stated in the following theorem (the proof is in Appendix C).

Theorem 6.3. *If $\mathcal{AM}$ satisfies Theorem* 6.2 *for any problem $DP = (SD, \mathcal{O})$ s.t. $\mathcal{O}$ satisfies $\mathcal{R}$, then $\mathcal{AM}$ is a correct mapping.*

Theorems 6.2 and 6.3 point out the essential role played by the notion of indiscriminability as introduced in this paper in order to assure the equivalence among the abstract diagnoses and the ground ones. In particular, indiscriminability is useful for reducing the

problem of deciding whether an abstraction fully preserves the diagnosis power to the problem of deciding whether a certain relation holds between all of the pairs of subsystem states that are merged by the abstraction. Deciding if such a relation holds may involve looking just at the model of one abstracted subsystem at a time (in case of local indiscriminability), looking at the whole system model (in case of global indiscriminability), or any possibility in between these two (in case of generic Γ^+-indiscriminability).

If we put together the above theorems with Theorem 5.1 (which states that all correct abstraction mappings are refinements of the abstraction mapping $\mathcal{AM}^G$ based on G-indiscriminability), we conclude that, once the partition of the system into subsystems has been determined, we need to adopt G-indiscriminability if we want to get the strongest abstraction that fully preserves ground diagnoses.

So far we have discussed the correspondence between abstract and ground diagnoses by assuming that the diagnostic algorithm computes all of the possible diagnoses. However, as discussed in Section 3, the notion of *preferred* diagnoses is important in practical diagnostic problem solving, where it is sometimes sufficient to focus on the diagnoses of minimum cardinality. The following theorem states an important result about the correspondence between ground and abstract preferred diagnoses, when a correct abstraction mapping is applied (the proof is reported in Appendix C).

Theorem 6.4. *Under the hypotheses of Theorem* 6.2, *let $\mathcal{PM}$ be a Preferred Abstraction Mapping derived from correct Abstraction Mapping $\mathcal{AM}$.*

For any ground state S_G that is a minimum cardinality diagnosis for the diagnostic problem DP involving N faults, the abstract state S_A obtained from S_G via $\mathcal{PM}$ is a preferred abstract diagnosis for DP_A with rank N. Moreover, for any S_A that is a preferred abstract diagnosis for DP_A with rank N, all of the states that can be obtained from S_A via the inverse preferred abstraction mapping $\mathcal{PM}^{-1}$ are minimum cardinality diagnoses for DP with N faults.

The theorem tells us that, if we are interested just in preferred diagnoses, computing the preferred diagnoses at the abstract level is equivalent to computing the minimum cardinality diagnoses at the ground level. One of the consequences is that the preferred abstract diagnosis of rank zero (i.e. the system behavior is nominal) is returned if and only if the preferred ground diagnosis assigns *ok* to all components: in this way, we can safely *detect* whether the system has (necessarily) a fault by computing the preferred diagnoses at the abstract level.

Note that this result is similar to the one obtained in [23] for the approach based on cones, but it is worth noting that the equivalence result between preferred diagnoses at the abstract and ground level shown in this paper concerns a much larger class of systems (non-directional systems vs directional ones) and allows the explicit representation of fault modes. The following example is aimed at clarifying these differences.

Example 6.1. Let us consider again the digital circuit depicted in Fig. 2, and the possible abstraction where the subsystem involving the logical gates $B1$, $B2$ and *NA* are aggregated into the abstract component NA^*. In Section 2, we have shown how the introduction of fault modes is disruptive for the abstraction method based on cones. In particular, we have shown that some of the preferred diagnoses can be lost.

We now show that these problems do not occur within our approach. By applying the notion of G-indiscriminability, it turns out that the abstract component NA^* has five abstract behavioral modes. In particular, the abstraction mapping is as follows:

$$\begin{aligned}
abm0 &= \{(ok, ok, ok)\} && \text{(rank 0)},\\
abm1 &= \{(ok, sa1, ok)\} && \text{(rank 1)},\\
abm2 &= \{(sa1, ok, ok)\} && \text{(rank 1)},\\
abm3 &= \{(sa1, sa1, ok)\} && \text{(rank 2)},\\
abm4 &= \{(ok, ok, sa1), (ok, sa1, sa1),\\
&\qquad (sa1, ok, sa1), (sa1, sa1, sa1)\}\\
&\qquad\qquad\qquad\qquad \text{(rank 1)}.
\end{aligned}$$

It is worth noting that each abstract behavioral mode is associated with a rank, which corresponds to the minimum rank among the indiscriminable ground states associated with the abstract behavioral mode. For example, $abm4$ has rank 1 even if some of the associated ground states have higher ranks (e.g., $(sa1, sa1, sa1)$ has rank 3).

If we restrict ourselves to the preferred mapping $\mathcal{PM}$ (which keeps only the preferred states in each set) we have that:

$$\begin{aligned}
abm0' &= \{(ok, ok, ok)\} && \text{(rank 0)},\\
abm1' &= \{(ok, sa1, ok)\} && \text{(rank 1)},
\end{aligned}$$

$$abm2' = \{(sa1, ok, ok)\} \quad \text{(rank 1)},$$
$$abm3' = \{(sa1, sa1, ok)\} \quad \text{(rank 2)},$$
$$abm4' = \{(ok, ok, sa1)\} \quad \text{(rank 1)}.$$

Let us reconsider the diagnostic problem involving the observations $I1(0)$, $I2(0)$, $O1(1)$, $O2(1)$. As we have already pointed out in Section 2, this diagnostic problem has two ground diagnoses of minimum cardinality:

$$G1 = \{B1(sa1), B2(sa1), NA(ok), N1(ok), N2(ok)\},$$
$$G2 = \{B1(ok), B2(ok), NA(ok), N1(sa1), N2(sa1)\}.$$

It is easy to verify that the above diagnostic problem also has two (preferred) abstract diagnoses of rank 2 when solved at the abstract level. In particular:

$$A1 = \{NA^*(abm3), N1(ok), N2(ok)\},$$
$$A2 = \{NA^*(ok), N1(sa1), N2(sa1)\}.$$

Abstract diagnosis $A1$ is mapped back by $\mathcal{PM}^{-1}$ to $G1$; in particular, $NA^*(abm3)$, is mapped back to $\{B1(sa1), B2(sa1), NA(ok)\}$. Similarly, abstract diagnosis $A2$ is mapped back to $G2$: indeed, $NA^*(ok)$ is mapped back to a nominal assignment $\{B1(ok), B2(ok), NA(ok)\}$ to the subcomponents of NA^*.

It is worth noting that the notion of rank plays a very important role in assuring the exact correspondence between abstract and ground diagnoses. In fact, if the rank of $abm3$ was set to 1 (i.e., one fault) instead of 2 in the abstract model, the abstract diagnosis process would have failed to recognize that diagnosis $A2$ is a preferred diagnosis.

7. Computing mappings and abstractions

7.1. Challenges

One of the main computational challenges of the abstraction task is the complexity of the search for the subsystems Γ of ground components to be mapped to abstract components AC_Γ. This problem can naturally be mitigated if we follow an incremental approach where we start by abstracting a small set of ground components (bound by a constant $\max_{merge}$) to derive an abstract component, which in turn can be abstracted with other ground or abstract components in a more abstract component, and so on.

In order to select the components to be merged we need to be heuristically guided by the *topology* of the system model, i.e. the graph whose set of vertices is the set of components C and whose set of edges contains edge (c_i, c_j) iff the component theories of c_i, c_j share at least one port. One possible approach consists in exploiting existing techniques such as graph partitioning and clique decomposition [31] or star-mesh transforms [30]. A simpler alternative adopted in this paper is to look for specific patterns in the topology graph (e.g., components connected in series/parallel).

Once the selection of the components to be abstracted has been solved, a second problem arises, concerning the *evaluation* of the abstraction mapping for the set of selected components: the abstraction should actually take place only if the mapping is *good enough*. What is a good enough abstraction? There is no simple answer to the question, but some criteria can be defined.

First of all, we consider that a mapping $\mathcal{AM}_\Gamma$ which defines an abstract component AC_Γ is *not* good if AC_Γ has a *large* number of behavioral modes compared to the number of behavioral modes of the components of the subsystem Γ abstracted in AC_Γ.[4] A high number of behavioral modes for component AC_Γ has at least two major, interrelated drawbacks: first of all, the component theory of AC_Γ could become too large and inefficient to store and manipulate; moreover, the cognitive burden put on the supervisor for understanding the behavior of AC_Γ could increase too much, contradicting the main purpose of abstraction which is to decrease such a burden.

One reasonable choice to address this issue, is to consider as acceptable an abstract component AC_Γ with a number of behavioral modes linear in $|\Gamma|$, i.e. such that $|dom(AC_\Gamma)| \leqslant \sum_{c\in\Gamma} \alpha_c \cdot |dom(c)|$; as a particular case, if we let $\alpha_c = 1$ for all c, this requirement becomes $|dom(AC_\Gamma)| \leqslant \sum_{c\in\Gamma} |dom(c)|$. This choice ensures that, no matter how many ground components we will be able to abstract into a single abstract component, the number of behavioral modes of such a component will grow at most linearly with the size of the abstracted subsystem. This limited complexity increase should then be much more than compensated by the reduction in the number of diagnoses that have

[4] Note that, in the worst case, the number $|dom(AC_\Gamma)|$ of abstract behavioral modes is $\prod_{c\in\Gamma} |dom(c)|$ (i.e., exponential in $|\Gamma|$).

to be computed and interpreted by the supervisor at the abstract level.

However, a limited number of abstract behavioral modes could not be sufficient to compactly represent the behavior of abstract component AC_Γ. In particular, the component theory of AC_Γ may still be too large because of the number of exogenous and endogenous ports. A measure which binds the size of such a relation from below is the number of exogenous ports $P_{exo}(\Gamma)$ of subsystem Γ (as pointed out, e.g., also in [20]); indeed, by definition of exogenous variable, all of the combinations of values of the exogenous ports are allowed for each behavioral mode of AC_Γ. We will make sure that the maximum number of exogenous ports is kept below the value of a constant $\max_{exo}$.

As for the endogenous ports, even when $|P_{end}(\Gamma)|$ is large, it may be possible that most of the combinations of the values of variables $P_{end}(\Gamma)$ do not appear in the domain theory of AC_Γ, simply because they are inconsistent with every behavior of the subsystem. Therefore, we don't enforce an upper bound on the number of such ports; however, we do enforce an upper bound $\max_{tuples}$ on the maximum number of tuples in the domain theory of AC_Γ.

Finally, a major computational problem concerns the partitioning of the states of subsystem Γ into indiscriminability classes, which is needed in order to ensure the correctness of the abstraction mapping. In this paper, we address this problem for *L*- and *G*-*indiscriminability*.

7.2. Abstraction algorithm

In order to compute abstractions, we propose algorithm *Abstract* shown in Fig. 6. The algorithm takes a system description *SD*, the system observability *O* and a context restriction $\mathcal{R}$, and returns an abstraction SD_A of *SD*.

The abstraction process is bootstrapped by computing wrapper mappings $\mathcal{AM}_{\{c\}}$ for each ground component $c \in C$ (with calls to function *LeafMapping*), and such mappings are applied to *SD*, producing a first abstraction SD_0 (lines 1–3).

A new Component Mapping $\mathcal{AM}_\Gamma$ is then computed by *ComponentMapping* before the main *while* loop; when no such mapping can be computed ($\mathcal{AM}_\Gamma = \emptyset$, see below), the loop terminates.

Within the body of the loop (lines 7–10), the mapping is applied to the previous system description SD_i to get a new abstraction SD_{i+1}. Function *ApplyMapping* simply computes the domain theory $DT(AC_\Gamma)$

Abstract($SD, O, \mathcal{R}$)
1 **foreach** $c \in C : \mathcal{AM}_{\{c\}} = \text{LeafMapping}(SD, O, \mathcal{R}, c)$
2 $SD_0 = SD$
3 **foreach** $c \in C : SD_0 = \text{ApplyMapping}(SD_0, \mathcal{AM}_{\{c\}})$
4 $i = 0$
5 $\mathcal{AM}_\Gamma = \text{ComponentMapping}(SD_i, O, \mathcal{R})$
6 **while** ($\mathcal{AM}_\Gamma \neq \emptyset$)
7 $SD_{i+1} = \text{ApplyMapping}(SD_i, \mathcal{AM}_\Gamma)$
8 **if** ($SD_{i+1} \neq \emptyset$)
9 $i = i + 1$
10 $\mathcal{AM}_\Gamma = \text{ComponentMapping}(SD_i, O, \mathcal{R})$
11 $SD_A = SD_i$
12 **return** SD_A

ComponentMapping($SD_i, O, \mathcal{R}$)
1 $\Gamma = \text{Choose}(SD_i, O, \mathcal{R}), 2 \leqslant |\Gamma| \leqslant \max_{merge}$
2 **if** ($\Gamma = \emptyset$) **return** $\emptyset$
3 $\mathcal{AM}_\Gamma = (AC_\Gamma, \Gamma, BM_\Gamma) = \text{Merge}(SD_i, O, \mathcal{R}, \Gamma)$
4 **if** ($|BM_\Gamma| > \sum_{c\in\Gamma} |dom(c)|$) **return** $\emptyset$
5 **return** $\mathcal{AM}_\Gamma$

Fig. 6. Algorithm `Abstract`.

of the new abstract component and then updates the sets of variables and component models in SD_i (see Section 4.2). As explained above, *ApplyMapping* also checks the number of tuples in $DT(AC_\Gamma)$, and, if it is larger than the bound max_{tuples}, it returns failure (i.e. $\emptyset$). In such a case, the index variable i is not incremented before the next call to *ComponentMapping*, i.e., we keep trying to find other admissible component abstractions for system description SD_i.

Function *ComponentMapping* (bottom of Fig. 6) computes the mapping $\mathcal{AM}_\Gamma$ of a subsystem Γ (with up to $\max_{merge}$ components) to an abstract component AC_Γ.

This computation can fail for two reasons: first, the choice of the subsystem Γ may itself fail (function *Choose*, see next section); second, the mapping $\mathcal{AM}_\Gamma$ may define more behavioral modes for AC_Γ than the sum of the behavioral modes of its subcomponents (line 4). In both of these cases, *ComponentMapping* returns $\emptyset$ instead of a valid mapping.

7.3. Choosing and merging sub-components

The choice of the (abstract) components $c_1, \ldots, c_k$ to be merged is guided by a heuristic based on the system topology. In this paper we give a simple definition of `Choose` that returns either $\emptyset$ (i.e., it was not possible to suggest a set of components to merge) or a pair of components to be merged, i.e., we let $\max_{merge} = 2$. Moreover, we restrict the choice to pairs of components that are either connected in series or in parallel

Choose(SD, O, $\mathcal{R}$)
1 **foreach** $c \in C$
2 **if** ($\exists c'$: in-series(SD, O, c, c') $\wedge$
3 $|P_{exo}(c)| + |P_{exo}(c')| \leqslant \max_{exo}$)
4 **return** (c, c')
5 **if** ($\exists c'$: in-parallel(SD, O, c, c') $\wedge$
6 $|P_{exo}(c)| + |P_{exo}(c')| \leqslant \max_{exo}$)
7 **return** (c, c')
8 **return** $\emptyset$

in-series(SD, O, c_i, c_j)
1 $\mathcal{N}_i$ = neighbors(SD, O, c_i)
2 $\mathcal{N}_j$ = neighbors(SD, O, c_j)
3 **if** $|\mathcal{N}_i| > 2$ **or** $|\mathcal{N}_j| > 2$ **return** *false*
4 **if** $c_j \notin \mathcal{N}_i$ **return** *false*
5 **return** *true*

in-parallel(SD, O, c_i, c_j)
1 $\mathcal{N}_i$ = neighbors(SD, O, c_i)
2 $\mathcal{N}_j$ = neighbors(SD, O, c_j)
3 **if** $|\mathcal{N}_i| \neq 2$ **or** $\mathcal{N}_i \neq \mathcal{N}_j$ **return** *false*
4 **return** *true*

Fig. 7. An example implementation of the `Choose` operator.

in the system topology. It is out of the scope of the present paper to study more general methods that can deal with topologies where the series/parallel pattern does not lead to significant abstractions because, e.g., the topology mostly contains complex structures, like *stars*. However, we note that, in such cases:

- it may be possible to adopt techniques from CSP and circuits analysis (such as graph partitioning and clique decomposition [31] or star-mesh transforms [30]) for suggesting components to be merged and/or for simplifying the system topology in order to make it more amenable to the series/parallel pattern,
- even when the topology still contains complex structures, it may be possible to simultaneously choose and merge more than two components at a time, looking particularly for sets of components that have a high number of connections among them (recall that Definition 4.1 of component abstraction mapping allows merging a bunch of components at once).

Function `Choose` (Fig. 7) considers each component c and checks whether it can be merged with another component c' s.t. c, c' are connected in series or in parallel; moreover, the sum of the number of exogenous ports in c, c' (which would become the ports of the abstract component) must not exceed the limit $\max_{exo}$. As soon as such a pair of components is found, `Choose` returns it; otherwise, $\emptyset$ is returned.

Merge(SD, O, $\mathcal{R}$, Γ)
1 $BM_\Gamma = \emptyset$
2 **foreach** $S_\Gamma = \{c_1(bm_1), \ldots, c_k(bm_k)\}$: $\lambda(S_\Gamma) = \{S_\Gamma\}$
3 **foreach** $S_\Gamma = \{c_1(bm_1), \ldots, c_k(bm_k)\}$ s.t. $\lambda(S_\Gamma) \neq \emptyset$
4 **foreach** $S'_\Gamma = \{c'_1(bm'_1), \ldots, c'_k(bm'_k)\}$
s.t. $S'_\Gamma \neq S_\Gamma$ and $\lambda(S'_\Gamma) \neq \emptyset$
5 **if** ($INTCHG = LOCAL$)
ind = LIndiscriminable(SD, O, $\mathcal{R}$, S_Γ, S'_Γ)
6 **if** ($INTCHG = GLOBAL$)
ind = GIndiscriminable(SD, O, $\mathcal{R}$, S_Γ, S'_Γ)
7 **if** (ind)
8 $\lambda(S_\Gamma) = \lambda(S_\Gamma) \cup \{S'_\Gamma\}$
9 $\lambda(S'_\Gamma) = \emptyset$
10 $BM_\Gamma = BM_\Gamma \cup \{(abm(S_\Gamma), \lambda(S_\Gamma))\}$
11 $\mathcal{AM}_\Gamma = (AC_\Gamma, \Gamma, BM_\Gamma)$
12 **return** $\mathcal{AM}_\Gamma$

Fig. 8. Function `Merge`.

Note that `Choose` must keep track of the pairs of components it returns, so that if the merge of such components fails, a different pair will be returned at the next call.

Predicate *in-series* (Fig. 7) requires that the components c_i and c_j have at most two neighbors, and that they are neighbors with each other (lines 3–4); as a consequence, the component resulting from merging c_i and c_j will also have at most two neighbors, i.e. it will not add complexity to the topology of the system. Note that operator `neighbors` considers as neighbors two components c_i and c_j if and only if they share at least one endogenous port and they do not share any observable in O. In this way, `Choose` never suggests the merge of two components if such a merge would result in the loss of some observable variable (according to the requirement in Definition 4.1).

Similarly, predicate *in-parallel* requires that the components c_i and c_j have exactly two neighbors, and that they share such neighbors (line 3); the component resulting from merging c_i and c_j will have exactly two neighbors and therefore, also in this case, no complexity is added to the topology of the system.

Function `Merge` (Fig. 8), given SD, O, $\mathcal{R}$ and a set Γ of components returned by `Choose`,[5] computes a mapping $\mathcal{AM}_\Gamma$. First of all (line 2), for each possible state $S_\Gamma = \{c_1(bm_1), \ldots, c_k(bm_k)\}$ of Γ, we initialize a set $\lambda(S_\Gamma)$ with $\{S_\Gamma\}$. Such a set is intended to represent the class of states indiscriminable from S_Γ.

Then, in the two nested loops (lines 3–10), we build the BM_Γ structure of mapping $\mathcal{AM}_\Gamma$. To this end, we

[5] Note that, although in our proposed implementation `Choose` returns pairs of components, the `Merge` algorithm can deal with a generic number k of subcomponents.

consider each pair of distinct states S_Γ and S'_Γ to check whether they are indiscriminable; the states whose associated set $\lambda(.)$ is empty are ignored because it means that they have already been added to some other indiscriminability class (see below).

States S_Γ and S'_Γ are checked either for L- or G-indiscriminability in lines 5–6, depending on constant *INTCHG*. For now, we just assume that the calls to *LIndiscriminable* and *GIndiscriminable* are functions that straightforwardly enforce the definitions of L- and G-indiscriminability; see the next two sections for more details.

If the two states turn out to be indiscriminable, S'_Γ is put in the indiscriminability class $\lambda(S_\Gamma)$ associated with S_Γ, and $\lambda(S'_\Gamma)$ is set to $\emptyset$ to indicate that S'_Γ should not be further considered in the next iterations of the algorithm; class $\lambda(S_\Gamma)$ will eventually become the definition of a new abstract behavioral mode $abm(S_\Gamma)$ in line 10, where the pair $(abm(S_\Gamma), \lambda(S_\Gamma))$ is added to BM_Γ.

Note that function *LeafMapping*, which is called in *Abstract* to compute the leaf mapping $\mathcal{AM}_{\{c\}}$ of a ground component c, is analogous to `Merge`, but it's much simpler since it has to deal just with component c instead of a set of components Γ (details on this simpler kind of abstraction can be found in [29]).

Now that we have detailed the functions `Choose` and `Merge`, it is possible to state a correctness result concerning the overall computation of abstractions (the proof is in Appendix C).

Theorem 7.1. *Let SD_A be the abstraction computed by algorithm* `Abstract` *given SD, O and $\mathcal{R}$, and let us assume that the set of abstract components C_A in SD_A is $\{AC_{\Gamma^1}, \ldots, AC_{\Gamma^p}\}$ (where AC_{Γ^i} aggregates the components of the ground subsystem Γ^i). Then, there exists an abstraction mapping $\mathcal{AM} = \{\mathcal{AM}_{\Gamma^1}, \ldots, \mathcal{AM}_{\Gamma^p}\}$ s.t.:*

- *$\mathcal{AM}$ is correct w.r.t. O, $\mathcal{R}$,*
- *SD_A is equal to the abstraction SD'_A obtained by applying $\mathcal{AM}$ to SD.*

The theorem has a straightforward but important corollary, which links the properties given in Section 6 with the computed abstraction SD_A.

Corollary 7.1. *Since the abstraction SD_A computed by* `Abstract` *is equivalent to an abstraction computed by applying a correct abstraction mapping to SD, all the important properties of Section 6 hold for SD_A.*

LIndiscriminable$(SD, O, \mathcal{R}, S_\Gamma, S'_\Gamma)$
1 $GDT(\Gamma) = DT_1 \bowtie \cdots \bowtie DT_k$
2 $\rho = \textbf{select}_{S_\Gamma \wedge \mathcal{R}} GDT(\Gamma)$
3 $\rho = \textbf{project}_{B(\Gamma)}(\rho)$
4 $\rho' = \textbf{select}_{S'_\Gamma \wedge \mathcal{R}} GDT(\Gamma)$
5 $\rho' = \textbf{project}_{B(\Gamma)}(\rho')$
6 **if** $(\rho \neq \rho')$ **return** *false*
7 **return** *true*

Fig. 9. Checking local indiscriminability.

7.4. Checking local indiscriminability

Checking the local indiscriminability of two states S_Γ, S'_Γ can be done with a function named `LIndiscriminable` (Fig. 9) which is a straightforward implementation of Definition 5.5.

First of all, we compute the global domain theory $GDT(\Gamma)$ of subsystem Γ (line 1). Then (lines 2–3), we derive a relation ρ which represents the possible combinations of values of the variables at the boundary $B(\Gamma)$ when the state of Γ is S_Γ; ρ is computed by simply selecting the tuples of $GDT(\Gamma)$ that agree with S_Γ and $\mathcal{R}$, and then by projecting such tuples on $B(\Gamma)$. A similar relation ρ' is computed w.r.t. state S'_Γ, and then ρ and ρ' are compared to decide whether S_Γ, S'_Γ are L-indiscriminable.

It is immediate to see that the time complexity of `LIndiscriminable` is $O(|DT_{\max}|^{|\Gamma|})$, where $|DT_{\max}| = \max_{c \in \Gamma} |DT_c|$; in particular, if $|DT_{\max}|$ is limited by the bound $\max_{tuples}$ (i.e. the maximum number of tuples allowed in the model of any ground or abstract component), and $|\Gamma|$ is limited by the bound $\max_{merge}$ (in our proposal $\max_{merge}$ is actually just 2), the time taken by the function has a constant upper bound.

7.5. Checking global indiscriminability

In principle, G-indiscriminability could be checked in the same way as L-indiscriminability. However, in the case of G-indiscriminability, the subsystem Γ is the whole set C of components, and $GDT(\Gamma)$ coincides with GDT.

Unfortunately, GDT can be a huge relation, and manipulating it directly can be prohibitive. In order to improve on this situation, we propose to encode GDT as an OBDD (Ordered Binary Decision Diagram). OBDDs [6] have been used in Artificial Intelligence to efficiently store and manipulate huge relations that would have been impossible to represent extensionally

GIndiscriminable($\mathcal{O}(GDT_{O,\mathcal{R}})$, S_Γ, S'_Γ)
1 $S_G = \textbf{\textit{gndrep}}(S_\Gamma)$
2 $\mathcal{O}(\rho) = \textbf{restrict}(\mathcal{O}(GDT_{O,\mathcal{R}}), S_G)$
3 $S'_G = \textbf{\textit{gndrep}}(S'_\Gamma)$
4 $\mathcal{O}(\rho') = \textbf{restrict}(\mathcal{O}(GDT_{O,\mathcal{R}}), S'_G)$
5 **if** ($\neg$**equiv**($\mathcal{O}(\rho)$, $\mathcal{O}(\rho')$)) **return** *false*
6 **return** *true*

Fig. 10. Checking global indiscriminability.

[3,8,16]; see Appendix B for a short summary on OBDDs and their operations.

The encoding of *GDT* as an OBDD $\mathcal{O}(GDT)$ is performed just once, before starting the whole abstraction process. In this phase, we also derive from $\mathcal{O}(GDT)$ another OBDD $\mathcal{O}(GDT_{O,\mathcal{R}})$ in order to take into account the observability O and context restriction $\mathcal{R}$:

$$\mathcal{O}(GDT_{O,\mathcal{R}}) = \mathcal{O}(GDT),$$
$$\mathcal{O}(GDT_{O,\mathcal{R}}) = \textbf{restrict}\big(\mathcal{O}(GDT_{O,\mathcal{R}}), \mathcal{R}\big),$$
$$\mathcal{O}(GDT_{O,\mathcal{R}}) = \textbf{remove}\big(\mathcal{O}(GDT_{O,\mathcal{R}}), P \cup I \backslash O\big).$$

Essentially, OBDD $\mathcal{O}(GDT_{O,\mathcal{R}})$ encodes the relation:

$$GDT_{O,\mathcal{R}} = \textbf{project}_{C \cup O}(\textbf{select}_{\mathcal{R}} GDT),$$

which contains the tuples of *GDT* that agree with the context restriction $\mathcal{R}$, and where the non-observable variables (except components) have been discarded.

Given $\mathcal{O}(GDT_{O,\mathcal{R}})$, any G-indiscriminability check between two states S_Γ, S'_Γ needed during the abstraction process can be performed with function `GIndiscriminable` (Fig. 10). Note that, when `GIndiscriminable` is called during the i-th iteration of the `Abstract` function (Fig. 6), Γ is a subsystem of abstract system SD_i and, therefore, S_Γ and S'_Γ are assignments to abstract components.

In order to check the G-indiscriminability of S_Γ and S'_Γ, we build two arbitrary ground states S_G and S'_G whose abstractions are, respectively, S_Γ and S'_Γ, and then we check the indiscriminability of S_G and S'_G at the ground level. The legitimacy of this reduction follows from the fact that all the ground states that map to S_Γ (resp. S'_Γ) are mutually G-indiscriminable, and therefore they are completely equivalent for performing a G-indiscriminability check between S_Γ and S'_Γ (see the proof of Theorem 7.1).

The actual computation of S_G and S'_G is performed by operator *gndrep*, which takes a state S_Γ of Γ and exploits the mappings $\mathcal{AM}_{\Gamma_0}, \ldots, \mathcal{AM}_{\Gamma_i}$ built so far by the `Abstract` function (see Fig. 6) in order to recursively find an arbitrary ground representative S_G of S_Γ. Note that operator *gndrep* has to traverse the hierarchy of mappings defined by $\mathcal{AM}_{\Gamma_0}, \ldots, \mathcal{AM}_{\Gamma_i}$ just once, and therefore it is very efficient.

Once the ground representative S_G of S_Γ has been built, we restrict $\mathcal{O}(GDT_{O,\mathcal{R}})$ with S_G, obtaining an OBDD $\mathcal{O}(\rho)$ which encodes the tuples of $GDT_{O,\mathcal{R}}$ that agree with S_G (line 2). Similarly, using the ground representative S'_G of S'_Γ, we obtain an OBDD $\mathcal{O}(\rho')$ which encodes the tuples of $GDT_{O,\mathcal{R}}$ that agree with S'_G (line 4).

The G-indiscriminability of S_G, S'_G reduces to checking the equivalence of OBDDs $\mathcal{O}(\rho)$ and $\mathcal{O}(\rho')$ (line 5). This follows immediately from the Definition 5.5 of indiscriminability.

Finally, it is important to note that the *restrict* and *equiv* operations performed by function `GIndiscriminable` take linear time w.r.t. the size of OBDD $\mathcal{O}(GDT_{O,\mathcal{R}})$ (Appendix B). Therefore, if the OBDD $\mathcal{O}(GDT_{O,\mathcal{R}})$ is of limited size, all of the calls to `GIndiscriminable` made during the abstraction process are efficient.

8. Experimental results

The aim of this section is to show that the abstraction of a non-trivial system model is not only feasible from a computational point of view, but also useful. In particular, we will explore experimentally the following points:

- to what extent the proposed abstraction methods are able to merge sets of ground components into abstract components?
- what is the impact of observability on the possibilities of making abstractions?
- are there significant differences in the abstractions that result from the adoption of local and global indiscriminability?
- is there a reduction in the number of preferred diagnoses when performing diagnostic reasoning at the abstract level rather than at the ground level?
- is there a reduction in the computational cost for getting the preferred abstract diagnoses instead of the preferred ground ones?

The first three points are relevant for evaluating the effectiveness of the method in building abstractions,

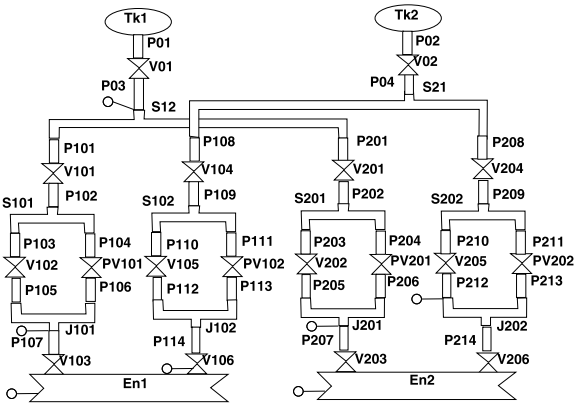

Fig. 11. The schema of the propulsion subsystem and the LOW observability.

while the latter two points are directly related with the benefits of using abstract models for diagnostic reasoning.

The experiments have been performed on a machine with an Intel Core 2 Duo 6600 CPU at 2.39 GHz and 4 GB of RAM, running the Linux Ubuntu 11.04 OS. All the algorithms are implemented in Java, and exploit the BuDDy C library for handling OBDDs.

As a test bed, we have selected the diagnosis of the propulsion system introduced in Example 4.1. We will consider two different levels of observability: the *HIGH* observability involving 17 endogenous observables (shown in Fig. 3) and the *LOW* observability involving just 7 endogenous observables (shown in Fig. 11). In both cases we have considered a (very) weak context restriction where the pyro-valves are commanded to be closed, i.e. $\mathcal{R}$ is equal to $\{cmd_{PV101}(close),\ cmd_{PV102}(close),\ cmd_{PV201}(close),\ cmd_{PV202}(close)\}$. The two levels of observability chosen for the experiments are intended to show the impact of a (significant) difference of observability on the abstraction. However, both of them have a solid motivation in terms of diagnostic reasoning. In particular, both *LOW* and *HIGH* observability are adequate for fault *detection*, i.e., they allow to detect the presence of any single fault given a suitable set of commands to the valves and pyro-valves. Moreover, they provide also a good degree of fault *localization*, since for all four major branches of the propulsion system there is at least a sensor able to detect whether some of the components belonging to that branch are not working properly.

The ablility to detect a fault and partially localizing it with the sensors output makes the *LOW* and *HIGH* observabilities realistic for a diagnostic scenario, where the exact localization and the *identification* of the specific fault(s) that have occurred can be postponed to further manual measurements and/or special test procedures. In particular, it is clear that the discriminative power offered by the *LOW* level of observability for fault identification is modest. As we will see in the rest of this section, the number of preferred diagnoses is quite large when the diagnostic reasoning is done on the ground model of the propulsion system. A higher discriminative power is provided by the *HIGH* level of observability which, however, is still quite limited, especially when we consider double faults.

We first show some results about the abstraction of the propulsion system obtained by considering the HIGH observability and by adopting the G-indiscriminability relation. The resulting abstraction of the propulsion system (shown on the left in Fig. 12) involves 23 abstract components.

It is easy to see that some abstract components (depicted in white) correspond to just a single ground component (e.g. tank $Tk1$ cannot be merged with any other component without causing the deletion of an observable variable), but most of the abstract components replace several ground components. For all of the abstract components, the number of behavioral modes is very low with respect to the number of states of the corresponding subsystems. For example, abstract component $P101 \rightarrow V101$ has 4 abstract behavioral modes, while $P107 \rightarrow En1$ and $S201 \rightarrow J201$ have respectively 5 and 8 modes. No abstract component has more than 8 modes.

Also the number of exogenous commands for each abstract component is low (maximum two). As explained in the paper, these characteristics of abstract components are essential in order to assure that the abstract model is described with relations of manageable size.

Intuitively a lower level of observability causes an increase of indiscriminability, and therefore an increase in the possibilities of abstraction. In Fig. 12 (right) we report the system resulting from abstracting the propulsion system with the *LOW* observability. It is apparent that the *LOW* level of observability induces stronger abstractions than the *HIGH* level (15 abstract components versus 23).

In Table 1 we compare the size of the component theory of some abstract components (in terms of number of tuples) with the size of $GDT(\Gamma)$ of the corresponding subsystems $\Gamma = \{c_1, \ldots, c_m\}$ (recall that $GDT(\Gamma)$ is the natural join of component models $DT_1, \ldots, DT_m$).

As one would expect, there are no savings when the abstract component coincides with a ground component, such as $J202$. However, the savings with the *HIGH* observability become significant as soon as the abstract component merges more components (see e.g. $P208 \rightarrow P209$) and become very relevant when the abstract component includes a more complex subsystem (see e.g. $S201 \rightarrow J201$).

Table 1

Size of the models of some abstract components (number of tuples) with respect to the size of the models of the corresponding subsystems

OBS	AbsComp	Abstract	Ground	Saving (%)
HIGH	$J202$	350	350	0.0
HIGH	$P102 \rightarrow J101$	1066	49,158	97.8
HIGH	$P107 \rightarrow En1$	160	576	72.2
HIGH	$S201 \rightarrow J201$	1082	17,850	93.9
HIGH	$P208 \rightarrow P209$	202	906	77.7
LOW	$P208 \rightarrow P209$	138	906	84.8
LOW	$P108 \rightarrow P114$	1764	2,532,402	99.9
LOW	$S202 \rightarrow P212_P213$	3230	71,754	95.5

The gains are even more impressive when we consider the *LOW* observability. An abstract component such as $P108 \rightarrow P114$, which corresponds to a quite large subsystem, can be represented in a very compact way with respect to such a subsystem (with a gain of 99.9%). It is also worth noting that the same abstract component e.g., $P208 \rightarrow P209$, can be represented in a more compact way in the case of *LOW* observability, since it has fewer abstract behavioral modes than in the case of *HIGH* observability (3 versus 5); the lower number of behavioral modes has also an impact on the size of the model of the abstract component (138 vs 202 tuples).

Table 2 summarizes the main characteristics of the abstractions obtained with *HIGH* and *LOW* observability. As expected, when we adopt global indiscriminability the impact of observability is quite relevant as concerns the number of abstract components and the average number of ground components abstracted into an abstract one. A parameter which deserves some

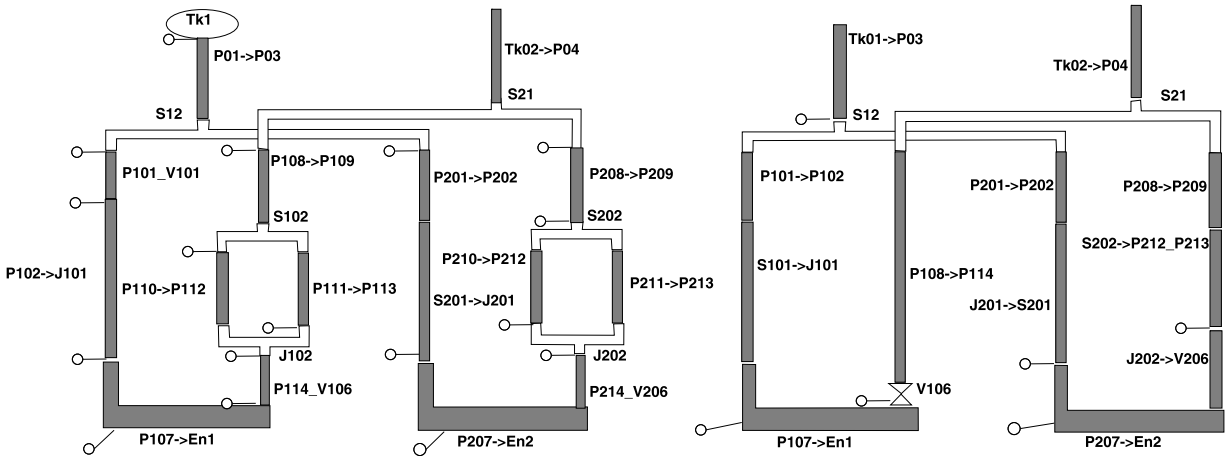

Fig. 12. Schema of the abstracted propulsion subsystem (HIGH and LOW).

Table 2

Comparing global and local indiscriminability with different levels of observability

	HIGH observability		LOW observability	
	G-ind	L-ind	G-ind	L-ind
# abs comps	23	30	15	28
# proper abs comps	16	19	13	18
avg # gnd comps	2.78	2.13	4.27	2.29
max # gnd comps	9	4	12	4
diag space	$7.68 * 10^{10}$	$3.48 * 10^{14}$	$3.24 * 10^{8}$	$9.66 * 10^{13}$
max # cmds	2	1	3	1
max # tuples	1082	350	3230	350

comments is the size of the diagnosis space (*diag space*), which represents the set of all possible diagnoses. As a reference measure, the size of the ground diagnosis space is $3.22 * 10^{22}$. The abstract diagnosis space is much smaller for abstraction with both *LOW* and *HIGH* observability. In particular, when we adopt G-indiscriminability, the lower level of observability is fully exploited and leads to a strong reduction (down to $3.24 * 10^{8}$).

Also the abstractions that adopt local indiscriminability are quite effective, although they are less powerful than the ones using G-indiscriminability when we compare them at the same level of observability. In particular, abstract components replace a lower number of ground components (e.g., 2.29 vs 4.27 for *LOW* observability). The same conclusion is true also for the size of the abstract diagnosis space; even more importantly, the local abstraction cannot fully exploit the potential of a low level of observability since, by definition, it exploits just the local context.

Overall, the results reported in Table 2 show that the abstraction process produces significant abstractions while the maximum size of abstract component models (*max # tuples*) is very reasonable. We now briefly comment on the time needed by the abstraction process. First of all, we note that it is not particularly relevant since abstraction is an off-line process. However, the abstraction methods we have implemented are also quite efficient. For example, the whole abstraction process takes less than 3 s of CPU time for abstracting the propulsion system when we adopt G-indiscriminability with *LOW* observability, and only slightly more time (about 3.3 s) with *HIGH* observability. It is worth noting that these results have been obtained by encoding the Domain Theory of the propulsion system with an OBDD, so that the check of G-indiscriminability can be done very efficiently.

So far we have shown that the abstraction is feasible and has the potential for providing significant benefits for diagnostic reasoning. In order to show that the reduction of the space of possible diagnoses has an actual impact on diagnostic reasoning, leading to a lower number of (abstract) diagnoses, we have generated two sets of diagnostic test cases.

The first set includes cases where a single fault has been injected and the values of the observables for the *HIGH* and *LOW* observability levels have been predicted with a software simulator. The test set is exhaustive for single faults, i.e. for each pipe we have injected the cl and br faults and for each valve we have injected the sc (when the valve is commanded open) and so (when the valve is commanded closed) faults. For experiments we are focusing our attention on the 78 cases where the fault is not masked by neither of the levels of observability. Note that fault masking is not due to *LOW* and *HIGH* observabilities, but to the context restriction that requires the pyro-valves to be closed. Indeed, all of the 16 cases when fault masking occurs correspond to faults in one of the sub-branches containing a closed pyro-valve and its two attached pipes.

The second set involves 496 cases; in each case, a double fault has been injected (typically the injected faults affect components in different branches of the propulsion system). All double fault cases were run with all the valves open, while all the pyro-valves were closed. Clearly, the capability of the diagnostic algorithm in singling out the diagnoses depends on the level of observability. With the *HIGH* observability, in 420 out of 496 cases the preferred ground diagnoses involve a double fault, whilst in 66 cases the preferred diagnoses involve just one fault (i.e. the double fault is masked). With the *LOW* observability the number of cases where masking does not occur drops to 366. In the following, we will refer to the test set involving all of the 496 cases as *DFA* (Double Fault ALL), while we will identify the subset involving just the 366 cases where the double fault is not masked as *DFNM* (Double Fault Not Masked).

For solving the test cases, we have run the diagnostic algorithm described in [27], which takes as input the domain model of the system to be diagnosed and compiles the global domain theory *GDT* to an OBDD. The algorithm computes an OBDD representing the whole set of diagnoses for a specific diagnostic problem, from which the minimum cardinality diagnoses are extracted in polynomial time.

We have run the diagnostic algorithm with five different models:

- the ground model of the propulsion system,
- the abstract model synthesized by considering G-indiscriminability and *HIGH* observability,
- the abstract model synthesized by considering L-indiscriminability and *HIGH* observability,
- the abstract model synthesized by considering G-indiscriminability and *LOW* observability,
- the abstract model synthesized by considering L-indiscriminability and *LOW* observability.

Since the different models involve a different number of components and behavioral modes, these variations have an impact on the size of the OBDD representing the compilation of the model itself. For example,

the size of the OBDD representing the ground model is 17085 nodes, whereas the OBDD representing the *GDT* of the abstract model based on *HIGH* observability and global abstraction involves 7376 nodes. All the OBDDs obtained by compiling the four abstract models mentioned above have a size lower than the OBDD encoding the ground model. The fact that the abstract models can be encoded in an OBDD of smaller size is a further demonstration that the abstraction captures some relevant structure in the model and encodes it efficiently.

We report the experimental data by comparing the number of preferred diagnoses (more specifically, the minimum cardinality diagnoses) returned at the ground and at the abstract level. We focus our attention on preferred diagnoses because these are the ones usually computed by diagnostic algorithms. More important, the main objective of our approach is to be informative for the human (or artificial) user of the diagnostic algorithm, and the set of all diagnoses (at least for the propulsion system) is not informative even when we consider abstract diagnoses (in all cases their number exceeds 10,000 even for single fault cases).

If one looks at the results reported in Table 3 concerning the test cases of single faults, it is apparent that abstraction has an effect in reducing the number of minimum cardinality diagnoses. In fact, even when we consider *HIGH* observability the number of preferred abstract diagnoses is roughly half of the preferred ground diagnoses. It is worth noting that there is a gain not only in the average number, but also in the maximum number of such diagnoses.

As expected, with *LOW* observability the number of ground preferred diagnoses is larger than with *HIGH* observability. The number of preferred abstract diagnoses obtained with global indiscriminability and *LOW* observability is not only a fraction of the corresponding ground diagnoses, but it is not too far from the number of abstract diagnoses obtained with *HIGH* observability. This is a further confirmation that global abstractions are able to fully exploit the reduction in observability. It is easy to see that also local abstractions yield significant benefits; however, these results confirm that local abstractions are not able to reduce the number of diagnoses to the same extent as global abstractions, particularly when the observability is *LOW*.

When we consider the results on double fault cases, the advantages of abstraction as concerns the number of preferred diagnoses become very relevant (see Table 4): in this table we report the data for the test set *DFA*, where some cases have a double fault masked by a single fault, as discussed above. The gain in terms of reduction of prefered diagnoses is impressive. Obviously, the largest gains are obtained when we consider *LOW* observability and G-indiscriminability, but in all of the scenarios the abstract preferred diagnoses are just a fraction of the corresponding ground diagnoses.

A similar pattern is observed when we consider the *DFNM* test set (see Table 5), where the double faults are never masked. Since the cases are more difficult to solve, the benefits of G-indiscriminability become even more apparent.

All these results show that the objective of reducing the number of diagnoses is achieved: there is a significant difference, from the point of view of the user, between inspecting and reasoning with (about) 5 abstract preferred diagnoses rather than with (about) 25 diag-

Table 3
Number of preferred diagnoses obtained with ground and abstract models for the test set of single fault cases

	HIGH observability			LOW observability		
	gnd	abs L-ind	abs G-ind	gnd	abs L-ind	abs G-ind
avg	4.62 ± 0.66	2.49 ± 0.23	2.23 ± 0.21	8.69 ± 1.07	5.29 ± 0.59	3.42 ± 0.55
min	1	1	1	1	1	1
max	8	4	4	13	8	7

Table 4
Number of preferred diagnoses obtained with ground and abstract models for double fault (*DFA* test set)

	HIGH observability			LOW observability		
	gnd	abs L-ind	abs G-ind	gnd	abs L-ind	abs G-ind
avg	21.34 ± 1.71	6.66 ± 0.34	5.09 ± 0.29	71.37 ± 5.16	27.16 ± 1.78	11.06 ± 0.99
min	2	1	1	5	3	1
max	64	16	12	156	56	40

Table 5
Number of preferred diagnoses obtained with ground and abstract models for double fault (*DFNM* test set)

	HIGH observability			LOW observability		
	gnd	abs L-ind	abs G-ind	gnd	abs L-ind	abs G-ind
avg	26.38 ± 2.06	7.89 ± 0.38	5.97 ± 0.32	94.94 ± 5.16	37.75 ± 1.69	14.63 ± 1.14
min	2	1	1	6	6	1
max	64	16	12	156	56	40

Table 6
CPU time (in ms) for solving single fault diagnostic cases with ground and abstract models

	HIGH observability			LOW observability		
	gnd	abs L-ind	abs G-ind	gnd	abs L-ind	abs G-ind
avg	75.01 ± 2.35	72.82 ± 2.45	66.82 ± 1.64	79.15 ± 3.10	74.32 ± 2.46	61.79 ± 2.00
max	97	114	80	100	105	75

Table 7
CPU time (in ms) for solving double fault diagnostic cases with ground and abstract models (*DFA* test set)

	HIGH observability			LOW observability		
	gnd	abs L-ind	abs G-ind	gnd	abs L-ind	abs G-ind
avg	176.00 ± 14.09	80.88 ± 1.38	60.28 ± 0.78	265.42 ± 15.93	138.10 ± 5.75	72.18 ± 2.30
max	688	189	139	869	249	164

Table 8
CPU time (in ms) for solving double fault diagnostic cases with ground and abstract models (*DFNM* test set)

	HIGH observability			LOW observability		
	gnd	abs L-ind	abs G-ind	gnd	abs L-ind	abs G-ind
avg	210.78 ± 15.25	84.30 ± 1.63	70.95 ± 0.96	337.39 ± 16.08	165.76 ± 5.48	78.79 ± 2.79
max	688	189	139	869	249	164

noses (or, in case of *LOW* observability, with 14 rather than over 90).

So far we have pointed out the benefits of abstraction as concerns the reduction of the number of preferred diagnoses, which was the main goal of our approach. However, it is worth noting that there is also a benefit from a computational point of view, since the CPU time required for solving a diagnostic case is lower for the abstract models than for the ground model.

Considering the data reported in Tables 6, 7 and 8, it is apparent that the CPU time for solving the diagnostic cases is low both for ground and abstract diagnoses. This depends on the properties of the diagnostic algorithm we have adopted, which (by exploiting the OBDD compilation) is able to solve all cases in a quite efficient way. In particular, it is worth noting that the max CPU times are low, and the max values are relatively close to the average values. Nevertheless, when abstract models are used, the computational cost is significantly reduced; for example, if we consider the *DFNM* cases, *LOW* observability and G-indiscriminability, such a cost is less than 25% of the cost of solving the diagnostic cases at the ground level.

9. Related work

Abstraction, intended as the ability to forget irrelevant details and to find simpler descriptions, has been studied and exploited in several fields of Computer Science (e.g., abstract interpretation [10], model checking [9]) and Artificial Intelligence (e.g., planning [21], theorem proving [14], perception [33]).

Although they deal with different domains and problems, all of such works are based on the definition of mappings (either built by humans or automatically) for transforming the domain models back and forth between the abstract and ground levels, as the Galois

connections $\langle \alpha, \gamma \rangle$ in abstract interpretation (in some cases, as in this paper, it is convenient to define the mapping γ from the abstract to the ground level as just the inverse α^{-1} of the mapping α from the ground to the abstract level). Moreover, in order to make effective use of abstractions, it is usually desirable to guarantee some form of correspondence between relevant entities or properties at the abstract and ground levels; for example, the safeness of a program semantics in abstract interpretation, the satisfaction of temporal logic formulas in model checking, the existence of a plan in planning, or the correctness of a theorem in theorem proving. These correspondences are analogous to the correspondence between abstract and ground diagnoses discussed above, and are expressed by properties analogous to the downward and upward failure properties studied in this paper (see, e.g., the definition of Theorem Decreasing and Theorem Increasing abstractions in [14]). It is important to point out that, in case of infinite domains, guaranteeing that the results obtained with an abstract representation are equivalent to the results obtained with the ground one may be undecidable (this is the case, e.g., for abstract interpretation [10]); in such cases, one may be satisfied with abstractions that, as the downward failure property discussed above, ensure that the abstraction is at least partially correct (e.g., in abstract interpretation, it does never produce false positives about the safeness of a program). On the other hand, if domains are finite (e.g., for the model checking discussed in [9]) it is usually possible to give such a guarantee, although it may be too computationally expensive or may lead to poor abstractions. In this paper, that also deals with finite domains, we have shown that full equivalence between abstract and ground diagnoses can be guaranteed with practically computable and useful model abstractions.

Focusing on previous approaches that specifically deal with abstraction in diagnosis, some of them have addressed the simpler problems of computing task-dependent abstractions by merging the values of single variables [22], or the behavioral modes of single components [29]. Compared to those works, the present work addresses the abstraction of component variables, which is in general a more complex task than the abstraction of the values of a single variable, since it requires to choose the set of component variables to be merged, to define the abstract behavioral modes of the abstract component in terms of the behaviors of the sub-components, and to update the model with the new component; all of these steps must be done carefully in order to obtain a useful abstraction that does indeed reduce the complexity of diagnostic reasoning and the size of the diagnostic results.

There exist a few previous works [19,28] that have addressed task-dependent component abstraction; however, compared to them, this paper covers for the first time a significantly more general class of systems, namely the ones that can be modeled with finite-domain relations. Contrary to the models considered by the previous works, relational models do not impose directionality between inputs and outputs of the system components, resulting in enhanced expressivity and reusability [24]. Moreover, in our models each component can be associated with a strong fault model for multiple behavioral modes.

Unlike hierarchical abstractions (e.g., [7,17,19,20, 23]), task-dependent abstractions produce abstract models that can completely replace the original models without incurring any loss of relevant diagnostic information. In particular, we have pointed out how existing hierarchical approaches usually guarantee the downward-failure property (i.e., no ground diagnosis is missed by using only the abstract model) but do not guarantee the upward-failure property (i.e., no spurious ground diagnoses are generated by using only the abstract model). This has two drawbacks compared with the task-dependent abstractions presented in this paper: first of all, in order to discard the spurious ground diagnoses, it is necessary to reason with the more fine-grained model(s); and, more importantly, the resulting diagnoses are expressed at the ground level. In fact, one of the main benefits of using our abstract models for diagnosis is that the result is a set of abstract diagnoses that are fewer and more informative than the ground ones.

Another important property of our abstractions is that they guarantee the equivalence between abstract- and ground-level diagnostic reasoning also for the computation of minimum cardinality diagnoses, which is often what the users need for practical purposes. When this property does not hold (as it is the case for most hierarchical approaches, e.g., [7,17,19]), even if we are interested just in minimum cardinality diagnoses at the ground level, all of the diagnoses at the abstract level must be computed, resulting in much higher computational costs. In this respect, a very interesting work is the hierarchical approach [23] based on cones, which guarantees the equivalence between minimum-cardinality abstract and ground diagnoses; however, contrary to our abstractions, the ones in [23] are restricted to directional system models (e.g., digital circuits) with weak fault models.

Finally, in the literature on diagnosis, there are works that aim to return the set of diagnoses in syn-

thetic form (e.g., kernel diagnoses [11] and scenarios [26]); diagnostic reasoning is performed on the ground model, and the main aim is to synthesize diagnoses for returning them in a more compact form. While such proposals share the goal of returning fewer, more informative diagnoses with the present paper, the purpose of our abstractions is that of deriving off-line abstract system models that allow the direct computation of abstract diagnoses. The two approaches may be seen as complementary, since further synthesis techniques may be applied to the abstract diagnoses computed with our method.

10. Conclusions

In the present paper we have presented a novel approach to automatic component abstraction for Model-Based Diagnosis. The main goal of the computed abstractions is to forget details which are irrelevant for the diagnostic purpose, given the current observability and operating conditions of the system; therefore, following [22], they can be classified as task-dependent abstractions.

Central to the properties of the abstractions described in this paper, is the notion of indiscriminability between pairs of states of a subsystem. Indiscriminability is essential for reducing the problem of deciding whether an abstraction fully preserves the diagnosis power to the problem of deciding whether a certain relation holds between all of the pairs of subsystem states that are merged by the abstraction. In particular, we have been able to show that computing abstractions based on indiscriminability is not only sufficient to guarantee the correspondence between abstract and ground diagnoses, but it is also necessary. Therefore, while abstractions that do not satisfy indiscriminability can still be useful for practical purposes (e.g., most hierarchical abstractions), they require a careful consideration of the consequences of the misalignment between abstract and ground diagnoses.

An interesting point is that there is a full spectrum of notions of indiscriminability, varying from local indiscriminability (which considers the abstracted subsystem without context) to global indiscriminability (which considers the whole system as the context). All of them guarantee the correspondence between abstract and ground diagnoses, but moving from local to global indiscriminability we get stronger abstractions that are more difficult to compute. Therefore, the choice of which notion to use can be viewed as a trade-off between the strength of the resulting abstraction and the computational effort to obtain it.

An important contribution of the paper is the description of algorithms for the actual computation of abstractions. The automatic synthesis of component abstractions is challenging since it not only requires to choose which ground components should be merged, but also to define the behavioral modes of abstract components, and to synthesize their models from the models of the ground components. We have introduced an algorithm able to compute abstractions based on the general notion of indiscriminability. Then, we have provided specialized functions for the important special cases of local and global indiscriminability.

As a final contribution, we have experimentally shown on a non trivial domain that the abstraction process is actually useful for solving diagnostic problems. The experiments have shown that the use of abstract models reduces the number of preferred diagnoses (with respect to the ground ones), exploits the level of observability of the system (when the observability is low, the gain in the number of preferred diagnoses is higher), and requires less time for computing diagnoses. Moreover, the experiments have confirmed that abstractions based on global indiscriminability are actually stronger (both in terms of reduction of the number of abstract diagnoses and of the computational effort) than abstractions based on local indiscriminability.

As an important direction of future work, we plan to study the abstractions based on notions of indiscriminability that fall in between the two limiting cases of local and global indiscriminability. Such notions can be useful when computing abstractions based on global indiscriminability becomes prohibitive because of the size and complexity of the system model; in such cases, we would like to fall back to a manageable notion of indiscriminability that, however, induces abstractions that are as strong as possible.

Appendix A. Component models for the propulsion system

Figure 13 reports the component models of a generic pipe, pump, junction, valve and sink, expressed as logical formulas over qualitative equations.

In our modeling approach, which is closely related to the one described in [24], we model the connectivity to source (i.e. a pump) and sink (i.e. an outlet of the system) for each component terminal. In particu-

domains	
$S \in \{src, snk\}$; $D \in \{in, out\}$	
$f_i \in \{-, 0, +\}$; $r_{S,D,i} \in \{+, \infty\}$	
general	
$(r_{S,in,i} = \infty) \wedge (r_{S,out,i} = \infty) \Rightarrow f_i = 0$	
$(r_{src,D,i} = \infty) \wedge (r_{snk,D,i} = \infty) \Rightarrow f_i = 0$	
pipe	$i, j \in \{1, 2\}, j \neq i$
pipe(*ok*)	$r_{S,in,i} = r_{S,out,j}; \quad f_i \oplus f_j = 0$
pipe(*cl*)	$r_{S,in,i} = \infty$
pipe(*br*)	$r_{snk,in,i} = +; r_{src,in,i} = \infty$
pump	$i \in \{1\}$
pump(*ok*)	$r_{src,in,i} = +; r_{snk,in,i} = \infty$
	$r_{snk,out,i} = \infty \vee f_i = -$
junction	$i, j, k \in \{1, 2, 3\}, j \neq i, k \neq i, j$
junction(*ok*)	$r_{snk,in,i} = \min(r_{snk,out,j}, r_{snk,out,k})$
	$r_{src,in,i} = \min(r_{src,out,j}, r_{src,out,k})$
	$f_i \oplus f_j \oplus f_k = 0$
valve	$i, j \in \{1, 2\}, j \neq i$
valve(*ok*)	*cmd*(*open*) $\Rightarrow$
	$r_{S,in,i} = r_{S,out,j}; \quad f_i \oplus f_j = 0$
	cmd(*close*) $\Rightarrow r_{S,in,i} = \infty$
valve(*so*)	$r_{S,in,i} = r_{S,out,j}; \quad f_i \oplus f_j = 0$
valve(*sc*)	$r_{S,in,i} = \infty$
sink	$i \in \{1\}$
sink(*ok*)	$r_{snk,in,i} = +; r_{src,in,i} = \infty$

Fig. 13. Models of hydraulic component types expressed with qualitative equations.

lar, given a component c and one of its terminals T_i^c, we define the following variables: $r^c_{src,in,i}$ (connectivity to source through c), $r^c_{src,out,i}$ (connectivity to source from outside c), $r^c_{snk,in,i}$ (connectivity to sink through c) and $r^c_{snk,out,i}$ (connectivity to sink from outside c). Such variables can take values $+$ (connection) and ∞ (no connection).

Beside connectivity, for each component terminal T_i^c we model the presence of flow and its direction with a variable f_i^c which can take values 0 (no flow), $+$ (flow enters c through T_i^c) and $-$ (flow leaves c through T_i^c).

Going back to the component models in Fig. 13, it is worth mentioning that we use qualitative equations and the $\oplus$ and $\otimes$ operators are the addition and multiplication in the sign algebra, while the min operator applied to connectivity variables takes value $+$ if at least one of its arguments has value $+$ and ∞ otherwise. The qualitative equations and the sign algebra operators they contain can be straightforwardly encoded as relations over discrete variables, as required by our Definition 3.1 of component description.

Apart from formulas related to the behavioral modes of the components, there are formulas representing background constraints; in particular, there are two general formulas which define situations where there can be no flow at a terminal T_i: the first formula applies when T_i is not connected to a source (or a sink) neither through c nor from the outside of c; the second formula applies when T_i is connected neither to a source nor to a sink through c (or from the outside of c).

Let us now comment the model of a pipe; the other models can be interpreted in a similar way. Pipes have two terminals T_1 and T_2. When the pipe is ok, the connectivity to source and sink propagates through the pipe and the qualitative sum of the flows at T_1, T_2 is 0 (i.e. either there is no flow, or the flow enters from one terminal and leaves from the other terminal). A clogged pipe (mode cl) propagates no connectivity. Finally a broken pipe (mode br) makes T_1 and T_2 behave as two sinks.

Appendix B. Ordered binary decision diagrams

An OBDD $\mathcal{O}(\mathcal{F})$ is a rooted DAG with (at most) two terminal nodes labeled **0** and **1** representing a Boolean function $\mathcal{F}(x_1, \ldots, x_n)$. The main reason to adopt OBDDs is that, compared to the representation of $\mathcal{F}$ as a truth-table, its OBDD representation can be exponentially smaller, since it exploits the structure of $\mathcal{F}$.

In order to build the OBDD $\mathcal{O}(\mathcal{F})$, it is necessary to specify a total order over the variables $x_1, \ldots, x_n$ of $\mathcal{F}$; given such a variable order, there is a unique OBDD of minimal size which encodes $\mathcal{F}$. The choice of the variable order is a very sensible issue, which can lead to exponentially different sizes of $\mathcal{O}(\mathcal{F})$, and has been deeply discussed in the literature (e.g. [1,4]).

When the Boolean function $\mathcal{F}$ has been encoded, several intractable tasks (such as model counting, test for consistency and, most notably, test for equivalence) become polynomial in the size of $\mathcal{O}(\mathcal{F})$; moreover, the Boolean functions encoded by one or more OBDDs can be efficiently manipulated with operators that act directly on OBDDs.

Table 9 summarizes some standard OBDD operators and their computational complexity: *apply* performs any binary logical operation on two functions, *restrict* assigns a constant value to a variable x_i, *equiv* tests two functions for equivalence and *remove* removes a vari-

Table 9
Standard OBDD operators and their complexity

op	Time	Output size
apply(op, $\mathcal{O}_1$, $\mathcal{O}_2$)	$O(\lvert\mathcal{O}_1\rvert \cdot \lvert\mathcal{O}_2\rvert)$	$\leqslant \lvert\mathcal{O}_1\rvert \cdot \lvert\mathcal{O}_2\rvert$
restrict($\mathcal{O}$, x_i), restrict($\mathcal{O}$, $\neg x_i$)	$O(\lvert\mathcal{O}\rvert)$	$\leqslant \lvert\mathcal{O}\rvert$
equiv($\mathcal{O}_1$, $\mathcal{O}_2$)	$O(\lvert\mathcal{O}_1\rvert + \lvert\mathcal{O}_2\rvert)$	N/A
remove($\mathcal{O}$, x_i)	$O(\lvert\mathcal{O}\rvert^2)$	$\leqslant O(2 \cdot \lvert\mathcal{O}\rvert)$

able by existentially quantifying it; in particular, the *remove* operator is equivalent to:

$$\mathcal{O} = \textbf{apply}\big(\vee, \textbf{restrict}(\mathcal{O}, x_i), \textbf{restrict}(\mathcal{O}, \neg x_i)\big).$$

A relation over multi-valued variables (such as the component and system models in this paper) can be encoded by an OBDD where each Boolean variable represents a specific value of a specific multi-valued variable, a conjunction of Boolean variables represents a tuple and the OBDD itself represents the characteristic function of the set of tuples in the relation.

Appendix C. Proofs

C.1. *Theorem* 5.1

We must prove that, if $\mathcal{AM}_\Gamma$ is correct w.r.t. O, $\mathcal{R}$, then it is a refinement of $\mathcal{AM}_\Gamma^G$. In other words, if two states S_Γ, S'_Γ are associated with the same abstract behavioral mode abm in $\mathcal{AM}_\Gamma$, then they must be associated with the same abstract behavioral mode abm^G in $\mathcal{AM}_\Gamma^G$.

Since $\mathcal{AM}_\Gamma$ is correct, it follows by definition that S_Γ, S'_Γ are Γ^+-indiscriminable for some $\Gamma^+ \supseteq \Gamma$. We must prove that this implies that S_Γ, S'_Γ are also G-indiscriminable, i.e., that $\mathcal{AM}_\Gamma^G$ associates them with the same abstract behavioral mode.

Let $\Delta^+ = \Gamma^+ \backslash \Gamma$, $\Delta^- = C \backslash \Gamma^+$, and $\Delta = \Delta^+ \cup \Delta^- = C \backslash \Gamma$. We assume, by contradiction, that states S_Γ, S'_Γ are Γ^+-indiscriminable, but *not* G-indiscriminable, i.e.:

$$\begin{aligned}&\textbf{project}_O(\textbf{select}_{S_\Gamma \wedge \mathcal{R} \wedge S_\Delta} GDT)\\ &\quad \neq \textbf{project}_O(\textbf{select}_{S'_\Gamma \wedge \mathcal{R} \wedge S_\Delta} GDT)\end{aligned} \tag{3}$$

for some $S_\Delta = S_{\Delta^+} \cup S_{\Delta^-}$. Let us consider a tuple containing S_Γ and S_Δ:

$$(S_\Gamma, S_{\Delta^+}, S_{\Delta^-}, \mathcal{X}, \mathcal{B}_{\Gamma^+}, \ldots, \mathcal{O}_{\Delta^-}) \in GDT, \tag{4}$$

where $\mathcal{X}$ is an instance of the (observable) exogenous ports, $\mathcal{B}_{\Gamma^+}$ is an instance of the obs-boundary $B(\Gamma^+)$ of Γ^+, and $\mathcal{O}_{\Delta^-}$ is an instance of the remaining observable variables.

We note that, by definition, *GDT* can be written as $GDT(\Gamma^+) \overset{B(\Gamma^+)}{\bowtie} GDT(\Delta^-)$, i.e. as the join between the model of Γ^+ and the model of Δ^- on the $B(\Gamma^+)$ variables. Therefore, tuple (4) above can be written as the join of a tuple in $GDT(\Gamma^+)$ and a tuple in $GDT(\Delta^-)$:

$$\begin{aligned}&(S_\Gamma, S_{\Delta^+}, \mathcal{X}_{\Gamma^+}, \mathcal{B}_{\Gamma^+}, \ldots)\\ &\quad \overset{B(\Gamma^+)}{\bowtie} (S_{\Delta^-}, \mathcal{X}_{\Delta^-}, \mathcal{B}_{\Gamma^+}, \ldots, \mathcal{O}_{\Delta^-}),\end{aligned} \tag{5}$$

where $\mathcal{X}_{\Gamma^+}$, $\mathcal{X}_{\Delta^-}$ partition $\mathcal{X}$ according to the variables associated, respectively, with Γ^+ and Δ^-.

Since S_Γ, S'_Γ are Γ^+-indiscriminable, $GDT(\Gamma^+)$ must also contain a tuple $(S'_\Gamma, S_{\Delta^+}, \mathcal{X}_{\Gamma^+}, \mathcal{B}_{\Gamma^+} \ldots)$. But then, such a tuple can be joined with the same tuple of $GDT(\Delta^-)$ used in (5):

$$\begin{aligned}&(S'_\Gamma, S_{\Delta^+}, \mathcal{X}_{\Gamma^+}, \mathcal{B}_{\Gamma^+}, \ldots)\\ &\quad \overset{B(\Gamma^+)}{\bowtie} (S_{\Delta^-}, \mathcal{X}_{\Delta^-}, \mathcal{B}_{\Gamma^+}, \ldots, \mathcal{O}_{\Delta^-})\end{aligned}$$

resulting in:

$$\big(S'_\Gamma, S_{\Delta^+}, S_{\Delta^-}, \mathcal{X}, \mathcal{B}_{\Gamma^+}, \ldots, \mathcal{O}_{\Delta^-}\big),$$

which, therefore, belongs to *GDT* and produces the same observables as tuple (4).

This means that tuple (4) cannot be the cause of the difference between the two parts of Eq. (3), i.e. of the G-discriminability of S_Γ, S'_Γ. We can apply the same argument to all the tuples of *GDT* that contain S'_Γ, concluding that S_Γ, S'_Γ are indeed G-indiscriminable. Contradiction.

C.2. *Theorem* 6.1

We prove that S_A is a diagnosis for $DP_A = (SD_A, \mathcal{O})$ iff there exists a diagnosis S_G for $DP = (SD, \mathcal{O})$ s.t. $S_A = \mathcal{AM}(S_G)$. Since we adopt a consistency-based definition of diagnosis, we can prove equivalently that a generic assignment S_A to the C_A variables is consistent with $GDT(SD_A)$, $\mathcal{O}$ iff there exists an assignment S_G to the C variables consistent with GDT, $\mathcal{O}$ s.t. $S_A = \mathcal{AM}(S_G)$.

Let us assume that $\mathcal{AM} = \{\mathcal{AM}_{\Gamma^1}, \ldots, \mathcal{AM}_{\Gamma^p}\}$ and that each $\mathcal{AM}_{\Gamma^i} = (AC_{\Gamma^i}, \Gamma^i, BM_{\Gamma^i})$ defines an

abstract component AC_{Γ^i}. Moreover, let us assume that each component AC_{Γ^i} has a domain $\{abm_{i,1}, \ldots, abm_{i,k_i}\}$ and that BM_{Γ^i} associates mode $abm_{i,j}$ ($j = 1, \ldots, k_i$) with a set $\lambda_{i,j}$ of ground states of subsystem Γ^i.

By construction (see Section 4.2), $AC_{\Gamma^i}(abm_{i,j})$ is consistent with an assignment $\mathcal{P}_{\Gamma^i}$ to the ports $P(\Gamma^i)$ in $DT(AC_{\Gamma^i})$ iff there is a state $S_{\Gamma^i} \in \lambda_{i,j}$ s.t. $AC_{\Gamma^i}(abm_{i,j}) = \mathcal{AM}_{\Gamma^i}(S_{\Gamma^i})$ and S_{Γ^i} is consistent with $\mathcal{P}_{\Gamma^i}$ in $GDT(\Gamma^i)$ (note that, since Γ^i has been abstracted into AC_{Γ^i} with a component abstraction mapping, all of the observables are endogenous ports of the subsystem, i.e. $O(\Gamma^i) \subseteq P(\Gamma^i)$).

Similarly, an assignment $S_A = \{AC_{\Gamma^1}(abm_{1,j_1}), \ldots, AC_{\Gamma^p}(abm_{p,j_p})\}$ is consistent with an assignment $\mathcal{P}_{1,\ldots,p} = \bigcup_{i=1,\ldots,p} \mathcal{P}_{\Gamma^i}$ to the variables $P_{1,\ldots,p} = \bigcup_{i=1,\ldots,p} P(\Gamma^i)$ in $GDT(SD_A)$ iff there is a ground state S_G s.t. $S_A = \mathcal{AM}(S_G)$ and S_G is consistent with $\mathcal{P}_{1,\ldots,p}$ in GDT.

Now, since the parameter $\mathcal{O}$ of a diagnostic problem DP is an assignment to the O variables which are a subset of the set of variables $P_{1,\ldots,p}$, we have proved that S_A is an abstract diagnosis iff there is some ground diagnosis S_G s.t. $S_A = \mathcal{AM}(S_G)$.

C.3. *Theorem* 6.2

From the proof of Theorem 6.1, we know that S_A is a diagnosis for DP_A *only if* there exists a diagnosis S_G for $DP = (SD, \mathcal{O})$ s.t. $S_A = \mathcal{AM}(S_G)$.

We still have to prove that, if S_A is a diagnosis for $DP_A = (SD_A, \mathcal{O})$, *all* the S_G s.t. $S_A = \mathcal{AM}(S_G)$ are ground diagnoses. First of all we note that, if S_G is a diagnosis for DP, then any other ground system state S'_G s.t. $\mathcal{AM}^G(S'_G) = \mathcal{AM}^G(S_G)$ is a diagnosis for DP, because S_G, S'_G are mapped to the same abstract state only when they are G-indiscriminable.

From the fact that $\mathcal{AM}$, by being correct, must be a refinement of $\mathcal{AM}^G$ (Theorem 5.1), it follows that if S_G is a diagnosis for DP, then any other ground system state S'_G s.t. $\mathcal{AM}(S'_G) = \mathcal{AM}(S_G)$ is a diagnosis for DP.

Now, let S_A be an abstract diagnosis for DP_A. If, for some $S_G \in \mathcal{AM}^{-1}(S_A)$, S_G is not a diagnosis for DP, then there is no diagnosis S'_G for DP s.t. $S_A = \mathcal{AM}(S'_G)$, which contradicts what we have stated at the beginning of this proof. It follows that all the assignments in $\mathcal{AM}^{-1}(S_A)$ are diagnoses for DP and, consequently, that the set of diagnoses for DP is the grounding of the set of diagnoses for DP_A.

C.4. *Theorem* 6.3

Let us prove that, if $\mathcal{AM}$ is *not* correct, then, for some problem $DP = (SD, \mathcal{O})$ s.t. $\mathcal{O}$ satisfies $\mathcal{R}$, $\mathcal{AM}$ does *not* satisfy Theorem 6.2, i.e., there exists an abstract state S_A which is a diagnosis for DP_A s.t. not all of the groundings $S_G \in \mathcal{AM}^{-1}(S_A)$ are diagnoses for DP.

We note that, if $\mathcal{AM}$ is *not* correct, it must violate in particular G-indiscriminability (Theorem 5.1). Let us consider $AC_\Gamma \in C_A$, and let S_Γ, S'_Γ be two states associated with $abm \in dom(AC_\Gamma)$ s.t. they are not G-indiscriminable, i.e.:

$$\mathbf{project}_O(\mathbf{select}_{S_\Gamma \wedge \mathcal{R} \wedge S_\Delta} GDT) \neq \mathbf{project}_O(\mathbf{select}_{S'_\Gamma \wedge \mathcal{R} \wedge S_\Delta} GDT).$$

Then, there exists a tuple $(S_\Gamma, S_\Delta, \ldots, \mathcal{O}) \in GDT$ for which no corresponding tuple $(S'_\Gamma, S_\Delta, \ldots, \mathcal{O})$ exists in GDT.

Given diagnostic problem $DP = (SD, \mathcal{O})$, $S_\Gamma \cup S_\Delta$ is a (ground) diagnosis for DP while $S'_\Gamma \cup S_\Delta$ is not; and, according to Theorem 6.1, $S_A = \mathcal{AM}(S_\Gamma \cup S_\Delta) = \mathcal{AM}(S'_\Gamma \cup S_\Delta)$ is an abstract diagnosis for DP_A whose grounding $S'_\Gamma \cup S_\Delta$ is not a diagnosis for DP.

C.5. *Theorem* 6.4

From the definition of the rank of an abstract behavioral mode (Definition 4.2) and the definition of the rank of a diagnosis (Section 3), it is easy to see that, if S_A is an abstract state, then the rank $r(S_A)$ of S_A is $\min(\{r(S_G)\colon S_G \in \mathcal{AM}^{-1}(S_A)\})$.

Therefore, if S_G is a preferred ground diagnosis for DP with cardinality N (i.e., with rank N), $S_A = \mathcal{AM}(S_G)$ has also rank N; in particular, S_G is one of the preferred states in $\mathcal{AM}^{-1}(S_A)$ and therefore the preferred component abstraction mapping $\mathcal{PM}$ is defined on S_G as $S_A = \mathcal{PM}(S_G)$ (Definition 4.2). Moreover, from Theorem 6.1 we know that S_A is an abstract diagnosis for DP_A. Since each abstract diagnosis must be the abstraction of at least one ground diagnosis (see the proof of Theorem 6.1), none of the other abstract diagnoses S'_A can have a rank $N' < N$; therefore, $S_A = \mathcal{PM}(S_G)$ is a preferred abstract diagnosis.

In the other direction, if S_A is a preferred abstract diagnosis for DP_A with rank N, then $\mathcal{PM}^{-1}(S_A)$ contains one or more ground states S_G of rank N. Moreover, from Theorem 6.2 we know that each $S_G \in$

$\mathcal{AM}^{-1}(S_A)$ is a ground diagnosis for *DP*. Since each ground diagnosis must be the grounding of an abstract diagnosis (see Theorem 6.1), none of the other ground diagnoses S'_G can have a rank $N' < N$; therefore, all of the states $S_G \in \mathcal{PM}^{-1}(S_A)$ are preferred ground diagnoses of cardinality N.

C.6. Theorem 7.1

First of all, let us prove by induction the correctness of a single call to *Merge* (which returns a mapping $\mathcal{AM}_\Gamma$) followed by the application of $\mathcal{AM}_\Gamma$ with *ApplyMapping*. More precisely, let us assume that the thesis of Theorem 7.1 holds before a call $\mathcal{AM}_\Gamma = \mathit{Merge}(\mathit{SD}_i, O, \mathcal{R})$ in line 3 of *ComponentMapping*, and let us prove that the thesis continues to hold after the subsequent call $\mathit{SD}_{i+1} = \mathit{ApplyMapping}(\mathit{SD}_i, \mathcal{AM}_\Gamma)$ in line 7 of *Abstract* (see Fig. 6). Therefore, we assume that, before the call to *Merge*, there exists an abstraction mapping $\mathcal{AM} = \{\mathcal{AM}_{\Gamma^1}, \ldots, \mathcal{AM}_{\Gamma^p}\}$ s.t.:

- $\mathcal{AM}$ is correct w.r.t. O, $\mathcal{R}$,
- SD_i is equal to the abstraction obtained by applying $\mathcal{AM}$ to *SD*,

where SD_i is the last abstract system description computed by *Abstract* so far.

The base of the induction is given by the initialization outside of the main *while* loop in function *Abstract*, where a first abstraction SD_0 is computed by wrapping the ground components into abstract components by applying the mappings $\mathcal{AM}_c$ computed by function *LeafMapping*. We assume the correctness of this initial abstraction (details can be found in [29]).

We prove the theorem by assuming that *Merge* adopts the notion of G-indiscriminability (i.e., *INTCHG* = *GLOBAL*). The proof is analogous for L-indiscriminability (and any other kind of Γ^+-indiscriminability).

Function *Merge* (Figure 8) considers each pair of states S_Γ, S'_Γ of (abstract) subsystem $\Gamma = \{\mathit{AC}_1, \ldots, \mathit{AC}_k\} \subseteq C_i$, where C_i is the set of (abstract) components belonging to SD_i, and puts them into the same class λ, associated with an abstract behavioral mode *abm* of the new abstract component AC, if and only if they are G-indiscriminable.

The components $\{\mathit{AC}_1, \ldots, \mathit{AC}_k\}$ in Γ are defined by a corresponding sub-mapping $\mathcal{AM}_i$ of $\mathcal{AM}$ s.t. $\mathcal{AM}_i = \{\mathcal{AM}_{\Gamma^{i,1}}, \ldots, \mathcal{AM}_{\Gamma^{i,k}}\} \subseteq \mathcal{AM}$. Let S_G (resp. S'_G) be a ground representative of S_Γ (resp. S'_Γ) according to $\mathcal{AM}_i$, i.e. $S_\Gamma = \mathcal{AM}_i(S_G)$ (resp. $S'_\Gamma = \mathcal{AM}_i(S'_G)$). Note that S_G, S'_G are states of a ground subsystem $\Gamma_G = \Gamma^{i,1} \cup \cdots \cup \Gamma^{i,k}$.

It is possible to show that states S_G and S'_G are G-indiscriminable iff S_Γ, S'_Γ are G-indiscriminable. Indeed, by construction of the abstract system description SD_i (see Section 4.2), it is easy to see that if there exists a tuple $(S_G, \ldots, \mathcal{O}) \in \mathit{GDT}$ (where $\mathcal{O}$ is an instance of the observable variables), there must be a tuple $(S_\Gamma, \ldots, \mathcal{O}) \in \mathit{GDT}_i$ (and similarly for S'_G and S'_Γ); moreover, since $\mathcal{AM}_i$ is correct (i.e., it maps ground states to the same abstract state iff they are G-indiscriminable), it also holds that, if there exists a tuple $(S_\Gamma, \ldots, \mathcal{O}) \in \mathit{GDT}_i$, then there will be a tuple $(S_G, \ldots, \mathcal{O}) \in \mathit{GDT}$ (and similarly for S'_Γ and S'_G). It follows that S_G and S'_G are G-indiscriminable iff S_Γ, S'_Γ are G-indiscriminable.

After *Merge* returns mapping $\mathcal{AM}_\Gamma$, *ApplyMapping* computes a new abstraction SD_{i+1} by applying $\mathcal{AM}_\Gamma$ to SD_i; in the new abstraction SD_{i+1}, states S_Γ and S'_Γ will be merged into some abstract behavioral mode abm of a new abstract component AC of SD_{i+1}. Then, the new mapping $\mathcal{AM}$ that satisfies the thesis of Theorem 7.1 must simply associate all of the states of subsystem Γ_G that belong to either $\mathcal{AM}_i^{-1}(S_\Gamma)$ *or* $\mathcal{AM}_i^{-1}(S'_\Gamma)$ with abm. It is easy to see that in this case:

- $\mathcal{AM}$ is still correct w.r.t. O, $\mathcal{R}$,
- SD_{i+1} is equal to the abstraction obtained by applying $\mathcal{AM}$ to *SD*,

i.e., the thesis of Theorem 7.1 continues to hold.

Given the correctness of one execution of *Merge* and *ApplyMapping*, the correctness of the whole *Abstract* algorithm follows straightforwardly from the fact that the execution of the algorithm essentially consists in a finite number of steps where, at each step:

- we either fail to find a suitable abstraction and go to the next step with no change to SD_i (this can be due, e.g., to the fact that the new abstract component had too many behavioral modes, or that its model had too many tuples), or
- we succeed, which means that:
  - we have chosen a subsystem Γ of two components s.t. none of its internal variables were observable,
  - we have successfully called *Merge* and *ApplyMapping* on subsystem Γ.

References

[1] F. Aloul, I. Markov and K. Sakallah, Faster SAT and smaller BDDs via common function structure, in: *Proc. ICCAD*, 2001, pp. 443–448.

[2] K. Autio and R. Reiter, Structural abstraction in model-based diagnosis, in: *Proc. ECAI98*, 1998, pp. 269–273.

[3] P. Bertoli, A. Cimatti, M. Roveri and P. Traverso, Planning in nondeterministic domains under partial observability via symbolic model checking, in: *Proc. IJCAI*, 2001, pp. 473–478.

[4] B. Bollig, M. Löbbing and I. Wegener, Simulated annealing to improve variable orderings for OBDDs, in: *Proc. ACM/IEEE IWLS*, 1995, pp. 5.1–5.10.

[5] F. Brglez and H. Fujiwara, A neutral netlist of 10 combinatorial benchmark circuits and a target translator in fortran, in: *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1985, pp. 104–111.

[6] R. Bryant, Symbolic Boolean manipulation with ordered binary-decision diagrams, *ACM Computing Surveys* **24** (1992), 293–318.

[7] L. Chittaro and R. Ranon, Hierarchical model-based diagnosis based on structural abstraction, *Artificial Intelligence* **155**(1,2) (2004), 147–182.

[8] A. Cimatti, C. Pecheur and R. Cavada, Formal verification of diagnosability via symbolic model checking, in: *Proc. IJCAI*, 2003, pp. 363–369.

[9] E.M. Clarke, O. Grumberg and D.E. Long, Model checking and abstraction, *ACM Trans. Program. Lang. Syst.* **16**(5) (1994), 1512–1542.

[10] P. Cousot, Abstract interpretation, *ACM Computing Surveys* **28**(2) (1996), 324–328.

[11] J. de Kleer, A. Mackworth and R. Reiter, Characterizing diagnoses and systems, *Artificial Intelligence* **56**(2,3) (1992), 197–222.

[12] O. Dressler and P. Struss, A toolbox integrating model-based diagnosability analysis and automated generation of diagnostics, in: *Proc. DX*, 2003, pp. 99–104.

[13] G. Friedrich, Theory diagnoses: A concise characterization of faulty systems, in: *Proc. IJCAI*, 1993, pp. 1466–1471.

[14] F. Giunchiglia and T. Walsh, A theory of abstraction, *Artificial Intelligence* **57**(2,3) (1992), 323–389.

[15] M. Goldszmidt and J. Pearl, Rank-based systems: a simple approach to belief revision, belief update and reasoning about evidence and actions, in: *Proc. KR92*, 1992, pp. 661–672.

[16] R.M. Jensen and M.M. Veloso, Obdd-based universal planning: Specifying and solving planning problems for synchronized agents in nondeterministic domains, *LNCS* **1600** (1999), 213–248.

[17] I. Mozetič, Hierarchical model-based diagnosis, *Int. J. Man–Machine Studies* **35**(3) (1991), 329–362.

[18] N. Muscettola, P. Nayak, B. Pell and B. Williams, Remote agent: to boldly go where no AI system has gone before, *Artificial Intelligence* **103** (1998), 5–47.

[19] D.J. Out, R. van Rikxoort and R. Bakker, On the construction of hierarchic models, *Annals of Mathematics and AI* **11** (1994), 283–296.

[20] G. Provan, Hierarchical model-based diagnosis, in: *Proc. DX*, 2001, pp. 167–174.

[21] E. Sacerdoti, Planning in a hierarchy of abstraction spaces, *Artificial Intelligence* **5** (1974), 115–135.

[22] M. Sachenbacher and P. Struss, Task-dependent qualitative domain abstraction, *Artificial Intelligence* **162**(1,2) (2005), 121–143.

[23] S. Siddiqi and J. Huang, Hierarchical diagnosis of multiple faults, in: *Proc. IJCAI*, 2007, pp. 581–586.

[24] P. Struss, A. Malik and M. Sachenbacher, Qualitative modeling is the key to automated diagnosis, in: *Proc. IFAC*, 1996.

[25] S. Subbarayan, Integrating csp decomposition techniques and bdds for compiling configuration problems, *Lecture Notes in Computer Science* **3524** (2005), 351–365.

[26] P. Torasso and G. Torta, Compact diagnoses representation in diagnostic problem solving, *Computational Intelligence* **21**(1) (2005), 27–68.

[27] P. Torasso and G. Torta, Model-based diagnosis through obdd compilation: a complexity analysis, *Lecture Notes in Computer Science* **4155** (2006), 287–305.

[28] G. Torta and P. Torasso, Automatic abstraction in component-based diagnosis driven by system observability, in *Proc. IJCAI*, 2003, pp. 394–400.

[29] G. Torta and P. Torasso, Parametric abstraction of behavioral modes for model-based diagnosis, *AI Communications* **22**(2) (2009), 73–96.

[30] M. van Lier and R. Otten, Planarization by transformation, *IEEE Transactions on Circuit Theory* **20**(2) (1973), 169–171.

[31] R. Weigel and B. Faltings, Compiling constraint satisfaction problems, *Artificial Intelligence* **115** (1999), 257–287.

[32] D.S. Weld and S. Addanki, Task-driven model abstraction, in: *Proc. Int. Workshop on Qualitative Physics*, 1990, pp. 16–30.

[33] J.-D. Zucker and L. Saitta, A model of abstraction in visual perception, *Applied Artificial Intelligence* **15**(8) (2001), 761–776.