Merging Indiscriminable Diagnoses: an Approach Based on Automatic Domains Abstraction

Pietro Torasso, Gianluca Torta
Dipartimento di Informatica
Università di Torino
Torino (Italy)
e-mail: {torasso,torta}@di.unito.it

Abstract. The paper presents an approach suitable for on-line diagnosis, which aims at automatically abstracting the domains of discrete variables in the model (i.e. behavioral modes of system components) in order to keep only those distinctions that are relevant given the available observations and their granularity.
In particular the paper describes an algorithm which identifies indistinguishable behavioral modes by taking into account specific classes of available observations and derives an abstract model where such modes are merged and the domain model is revised accordingly.
By considering increasingly restricted classes of available observations (and/or granularity of observations), a set of abstract models can be derived that can be exploited through model selection each time a new diagnostic problem has to be solved.
The approach has been tested within the framework of a diagnostic agent for a space robotic arm, and experimental results showing the reduction in the number of diagnoses are reported.

1 Introduction

Model based diagnosis has been applied successfully to automatic on-board diagnosis problems in a variety of domains, including automotive and space missions ([1], [10]).

While many problems are common to off-line and on-line diagnosis, the latter presents some peculiar challenges, the most apparent of which concerns the tough constraints on computational resources and time ([3]).

Another difficult problem both on-line and off-line diagnosis have to deal with is the potentially large number of alternative diagnoses returned by a diagnostic system when a specific problem has to be solved.
One classical way of addressing this problem consists in defining preference criteria among diagnoses, usually based on some form of minimality (see e.g. [6]) or probability, so that a number of admissible diagnoses can be discarded because of their implausibility .
We can also approach the problem not at diagnosis time, but at earlier time (i.e. during system design and modeling): there exist guidelines for creating models suitable for troubleshooting (see e.g. [8]) as well as methods for suggesting the placement of enough sensors in the system to guarantee that only one or a few admissible diagnoses will be returned in each situation (see for example [15]); sensors failures can be handled by an adequate level of redundancy.
Finally, the encoding of large sets of diagnoses in a compact way can at least alleviate the explosion of time and space required to compute and handle such large sets (see [11]).

Unfortunately, all these approaches only provide a partial solution; while preference criteria, cleverly written models and compact encoding do not guarantee that the reduced set of diagnoses is small enough in all situations, exhaustive sensor placement may be too expensive or just impossible because the device design is already frozen.

In off-line diagnosis, there's an additional possibility: when the number of diagnoses returned on the basis of available observations is too high, further discriminant measures can be automatically suggested and manually taken until a satisfactory level of discrimination is reached. Effective techniques based on information theory and probability have been devised to support this process (e.g. [7]). However, for on-board diagnosis, this approach is inadequate since in most cases the only available measures are provided by sensors and taking further measures manually is out of question.

In this paper we present an approach suitable for on-board diagnosis, which aims at automatically abstracting the domains of discrete variables in the model (i.e. behavioral modes of system components) in order to keep only those distinctions that are relevant given the available observations and their granularity. As we shall see, this can significantly reduce the number of returned diagnoses.

The paper is structured as follows. In section 2 we introduce some definitions, in particular the notion of indistinguishability among the behavioral modes of a component. In section 3 we present an algorithm which identifies indistinguishable behavioral modes by taking into account specific classes of available observations and derives an abstract model where such modes are merged and the domain model is revised accordingly. Section 4 discusses some ways the algorithm can be used effectively in diagnostic problem solving.

In section 5 we report experimental results obtained by implementing and running the algorithm on the model for a space robotic arm. Finally, in section 6 we briefly review other approaches in the literature and underline similarities and differences with respect to our own.

2 Basic Definitions

First, we define a system structure description (SSD) by slightly modifying the definition in [5]:

Definition 2.1 *A Structured System Description (SSD) is a tuple* $\langle V, \mathcal{G}, DT \rangle$ *where:*

- V *is a set of variables whose domains* $DOM(v), v \in V$ *are discrete and finite. Moreover, variables in* V *are partitioned in the following sorts: CXT (inputs), COMPS (components), STATES (endogenous variables), OBS (observables)* [1]
- DT *(Domain Theory) is a set of Horn clauses defined over* V *representing the behavior of the system (both normal and faulty). Note that the clauses are constructed in such a way that the roles associated with variables belonging to different sorts are respected:* CXT *and* $COMPS$ *variables will always appear in the body of clauses;* OBS *variables will always appear as heads of clauses;* $STATES$ *variables can appear in both*
- $\mathcal{G}$ *(System Structure) is a DAG whose nodes are in* V *representing the structure of the system. The graph can be directly computed from* DT*, being just a useful way for making explicit the structural properties "hidden" in* DT *clauses: whenever a formula* $N_1(bm_1) \wedge \ldots \wedge N_k(bm_k) \Rightarrow M(bm_l)$ *appears in* DT*, nodes* N_1 *through* N_k *are parents of* M *in the graph*

Since the system structure graph $\mathcal{G}$ is a DAG, a partial precedence relation holds between connected nodes in the graph:

Definition 2.2 *We denote with* $\succ$ *the usual precedence partial order relation over nodes in DAG* $\mathcal{G}$*, i.e.:* $N \succ M$ *if there exists a directed path from* N *to* M*.*

Given an SSD we can define specific diagnostic problems over it:

Definition 2.3 *A diagnostic problem is a tuple* $DP = \langle SDD, \mathbf{OBS}', \mathbf{CXT} \rangle$ *where SSD is the System Structured Description,* $\mathbf{OBS}'$ *is an instantiation of* $OBS' \subseteq OBS$ *and* $\mathbf{CXT}$ *is a complete instantiation of* CXT

We are now ready to give our definition of diagnosis, which is a fully abductive characterization [2] (see [4]):

Definition 2.4 *Given a diagnostic problem* $DP = \langle SDD, \mathbf{OBS}', \mathbf{CXT} \rangle$ *an assignment* $H = \{c_1(bm_1), \ldots, c_n(bm_n)\}$ *of a behavioral mode to each component* $c_i \in COMPS$ *is a diagnosis for DP if and only if:*

$$\forall m(x) \in \mathbf{OBS}' \quad DT \cup \mathbf{CXT} \cup H \vdash m(x)$$

and

$$\forall m(x) \in \mathbf{OBS}' \quad DT \cup \mathbf{CXT} \cup H \nvdash m(y) \text{ for } y \neq x$$

[1] We assume that observables never influence other variables. This is not restrictive: each observable parameter which influences other variables is modeled as an endogenous variable (i.e. it belongs to $STATES$) with an associated observable in OBS

[2] Note however that our approach does not depend on the definition of diagnosis being abductive vs consistency-based

Since in our definition, $\mathbf{OBS}'$ is (in general) a partial instantiation of OBS, we can introduce the notion of diagnoses that can't be discriminated given $\mathbf{OBS}'$ but that may be discriminated if more observables were available:

Definition 2.5 *Given a diagnostic problem* $DP = \langle SDD, \mathbf{OBS}', \mathbf{CXT} \rangle$*, let us suppose that* $H1$ *and* $H2$ *are two diagnoses for DP.* $H1$ *and* $H2$ *are discriminable if and only if* $\exists\ m \mid m \in (OBS - OBS')$ *such that*
$DT \cup \mathbf{CXT} \cup H1 \vdash m(a)$
$DT \cup \mathbf{CXT} \cup H2 \vdash m(b)$
$m(a) \neq m(b)$

Diagnoses are complete instantiations of variables in sort $COMPS$. We now turn into considering two such assignments $A1$ and $A2$ and compute the projections [3] of their transitive closures [4] over OBS ($\mathbf{OBS1} = project_{OBS}(tclosure(A1))$ and $\mathbf{OBS2} = project_{OBS}(tclosure(A2))$ respectively), given a fixed context $\mathbf{CXT}$.
If $\mathbf{OBS1} = \mathbf{OBS2}$ then $A1$ and $A2$ are indiscriminable diagnoses for diagnostic problem $\langle SDD, \mathbf{OBS}, \mathbf{CXT} \rangle$ where $\mathbf{OBS} = \mathbf{OBS1}$ (and $= \mathbf{OBS2}$). An interesting relation between $A1$ and $A2$ holds when this situation happens under *any* fixed context $\mathbf{CXT}$:

Definition 2.6 *Let* $A1$ *and* $A2$ *be two complete instantiations of* $COMPS$*; if, given any context* $\mathbf{CXT}$*,* $project_{OBS}(tclosure(A1)) = project_{OBS}(tclosure(A2))$*, then we say that* $A1$ *and* $A2$ *are indiscriminable.*

In the above definition we have considered the case where all OBS are available. Let's now consider the case (as it is usual in on-board diagnosis) when we can identify subsets of OBS that may be the only available manifestations (e.g. only sensorized manifestations may be available on-board, with no possibility to perform further measurements).
Let $\{CL_k\}$ denote such identified interesting subsets (not necessarily all disjoint); we can now refine definition 2.6 as follows:

Definition 2.7 *Two assignments* $A1$ *and* $A2$ *are* CL_k*-indiscriminable iff they are indiscriminable by considering* OBS *restricted to* CL_k*, i.e.* $\forall \mathbf{CXT}\ project_{CL_k}(tclosure(A1)) = project_{CL_k}(tclosure(A2))$

Given the above definitions, we are now ready to characterize formally two behavioral modes (i.e. values from the domain of a component variable c_i) that may be safely collapsed together without loosing any discriminability power of the model:

Definition 2.8 *Let* bm_r *and* bm_s *be two behavioral modes of component variable* c_i*; if for any two assignments* $A1 = (\alpha 1 \wedge c_i(bm_r) \wedge \alpha 2)$ *and* $A2 = (\alpha 1 \wedge c_i(bm_s) \wedge \alpha 2)$ *such that they differ only in the mode associated to* c_i*,* $A1$ *and* $A2$ *are* (CL_k*-)indiscriminable, then we say that* bm_r *and* bm_s *are* (CL_k*-)indistinguishable.*

[3] A projection of a set of instantiated variables $\mathbf{I}$ over a set of variables W ($project_W(\mathbf{I})$) is just the subset of $\mathbf{I}$ that mentions variables in W

[4] The transitive closure of Ai (tclosure(Ai)) is the set of $m(x)$ s.t. $DT \cup \mathbf{CXT} \cup Ai \vdash m(x)$

3 Automatic Domain Abstraction

3.1 The Algorithm

In this section we present an algorithm which identifies indistinguishable modes in a given model (that we will refer to as the *detailed* model), and generates a simplified model (that we will call *abstract*) where mutually indistinguishable modes are merged in new modes. The algorithm assumes that the model is defined as in definition 2.1 and further assumes that in the system structure graph $\mathcal{G}$ at most one directed path exists between any two nodes.

The top level function `Abstract()` is sketched as pseudo-code in figure 1 while other relevant functions called by `Abstract()` are showed in figure 2.

Parameter $CL_k \subseteq$ `Obs` of `Abstract()` contains the list of available manifestations, while Π_{CL_k} associates to each $M \in CL_k$ its granularity in the form of a partition Π_M over $DOM(M)$.
Manifestations that aren't available at all do not belong to CL_k. If M is available at a certain level of granularity, Π_M will contain as many classes as the distinguishable values for M, and each class will contain all the $v \in DOM(M)$ that can't be distinguished at the available level of granularity. As a special case, if M is available at its maximum granularity, Π_M will contain a separate class for each $v \in DOM(M)$.

The first few instructions of `Abstract()` perform an initial abstraction of the model based on Π_{CL_k}: indistinguishable values for each manifestation M (i.e. those that belong to the same class in Π_M) are substituted in `DT` by a new "abstract" value representing the whole class.

The call to `TopologicalSort()` returns a list containing variables in `Comps` $\cup$ `States` such that if two variables N, M satisfy relation 2.2 (i.e. $N \succ M$) we guarantee that $position(N) > position(M)$. In particular, we start a visit of the system structure graph $\mathcal{G}$ at the available observation nodes and proceed backwards by visiting a node only if all its immediate successors have already been visited.
Note that, by starting the visit at the available manifestations only (i.e. CL_k), some of the `Comps` and `States` may not be reached at all; these nodes, that are connected only to unavailable manifestations, are stored in a `TrivialNodes` list (see below).

The main loop in `Abstract()`, for each variable N in the list, first computes the conditions under which the variable influences its immediate successors modes (this is recorded in an associative memory `InfluencesMatrix[]`); then, by using `InfluencesMatrix[]` it computes the partition of all the modes of the variable in equivalence classes determined by the indistinguishability relation (`FindIndistinguishableModes()`); finally it replaces the occurences of the modes in the `DT` clauses with newly introduced "class representative" modes (`MergeModes()`).
If the call to `FindIndistinguishableModes()` produced a trivial partition for N (i.e. only one class coinciding with $DOM(N)$) then N itself is added to the list of trivial nodes.

When `Abstract()` terminates, `TrivialNodes` contains the components and states whose behavioral modes are all equivalent in influencing relevant manifestations (i.e. $M \in CL_k$). These nodes, together with unavailable manifestations (i.e. $M \in$ `Obs`$\backslash CL_k$) are obviously redundant for the diagnostic task and the caller of `Abstract()` may decide to completely remove them from the model.

Let's now describe into some more detail the functions called by `Abstract()` (figure 2).
Function `FindInfluences()` considers how each mode bm_r of variable N under consideration can cause mode bm_s of immediate successor variable M. The condition under which $N(bm_r)$ causes $M(bm_s)$ is clearly the disjunction of conjunctions of the form $\alpha = \alpha_1 \wedge \alpha_2$ where α_1 and α_2 occur in a formula $\alpha_1 \wedge N(bm_r) \wedge \alpha_2 \Rightarrow M(bm_s)$.
Function `FindIndistinguishableModes()` is recursive; at each call it partitions a set of modes into indistinguishability classes based on a single immediate successor node and then calls itself recursively on each of the generated equivalence classes in order to further discriminate by considering the remaining immediate successors.
Note that in the test $(\langle \alpha, \pi \rangle \in \Pi_{cond})$ we are testing propositional formulas for identity; we assume that any two equivalent formulas have been made identical at that point by calls to *normalize()* in `FindInfluences()`. Normalization is not too computationally expensive since the formulas we handle are in DNF and only positive literals can occur.

Function `MergeModes()`, given a partition Π (either an element of Π_{CL_k} or computed by `FindIndistinguishableModes()`), considers the equivalence classes π one at a time. It generates a new name ν as a "representative" for the class and then scans the `DT` set of formulas for occurrences of $bm \in \pi$ and replaces them with ν. This process can produce duplicate formulas [5]; by using set notation in the pseudo-code we underline that only one copy of the duplicate formulas has to be added to the new version of `DT`.

3.2 Correctness

In this paragraph we state two properties which imply that the abstraction algorithm behaves as intended.

Property 3.1 *If two behavioral modes are put in the same class π by function* `FindIndistinguishableModes()` *they are CL_k-indistinguishable in the sense of definition 2.8.*

Proof. Given assignments $A1 = \alpha_1 \cup \{N(bm_{r1})\} \cup \alpha_2$ and $A2 = \alpha_1 \cup \{N(bm_{r2})\} \cup \alpha_2$ suppose $DT \cup \mathbf{CXT} \cup A1 \vdash m(x)$ while $DT \cup \mathbf{CXT} \cup A2 \not\vdash m(x)$ for some $m \in CL_k$. Clearly, it can't be $m(x) \in tclosure(\alpha 1 \cup \alpha 2)$ because otherwise $m(x)$ would be derivable from $A2$ as well.
Then, the entailment of $m(x)$ by $A1$ must exploit at some point $N(bm_{r1})$ by using a formula $\varphi = (N(bm_{r1}) \wedge \gamma \Rightarrow L)$.
If $L = m(x)$, i.e. the formula directly entails $m(x)$, then, an analogous formula $\varphi' = (N(bm_{r2}) \wedge \gamma' \Rightarrow m(x))$ must exist in DT, with $\gamma \Leftrightarrow \gamma'$ (indeed, two modes are put in the same partition only if they have the same direct effects under the same conditions). Then, $DT \cup \mathbf{CXT} \cup A2 \vdash m(x)$, which is a contradiction.
This result can be extended to the case when $L \neq m(x)$ (i.e. the number of steps between the application of formula φ and the conclusion $m(x)$ is greater than 1) with a proof by induction. □

[5] This is not incidental: the value of our abstraction partially lies in the collapse of formulas

```
Function Abstract(V = ⟨ Cxt, Comps, States, Obs ⟩, G, DT, CL_k, Π_{CL_k})
     ForEach M ∈ CL_k
          DT := MergeModes(M, Π_{CL_k}(M), DT)
     Loop
     Candidates := TopologicalSort(States ∪ Comps, CL_k, G)
     TrivialNodes := States ∪ Comps \ Candidates
     InfluenceMatrix := ∅ × ∅ × ∅
     ForEach (N ∈ Candidates)
          ImmediateSuccessors := {children of N in the system structure graph G} ∩ (Canditates ∪ CL_k)
          InfluenceMatrix := InfluenceMatrix ∪ FindInfluences(N, ImmediateSuccessors)
          Π := FindIndistinguishableModes(N, modes(N), ImmediateSuccessors, InfluenceMatrix)
          If (Π = {DOM(N)}) Then TrivialNodes := TrivialNodes ∪ {N}
          DT := MergeModes(N, Π, DT)
     Loop
     Return
EndFunction
```

Figure 1. Sketch of the `Abstract()` function

The following property is intended to demonstrate the correspondence of a diagnosis at the abstract level to a set of diagnoses at the detailed level.

Property 3.2 *Let* $DP_d = \langle SSD_d, \mathbf{OBS}', \mathbf{CXT} \rangle$ *be a diagnostic problem and* $DP_a = \langle SSD_a, \mathbf{OBS}', \mathbf{CXT} \rangle$ *the corresponding problem at the abstract level. Then,* $D_a = \{c_1(\nu_1), \ldots, c_n(\nu_n)\}$ *where* ν_i *is a new behavioral mode introduced in place of set* $\{bm_{i1}, \ldots, bm_{ik_i}\}$ *of indistinguishable behavioral modes is a diagnosis for* DP_a *iff all the elements in the set:*
$\{\{c_1(bm_{1j_1}), \ldots, c_n(bm_{nj_n})\}, j_i = 1 \ldots k_i\}$
are diagnoses for DP_d.

Proof. Our proof is subdivided in 3 steps: first, we prove that for any two diagnoses at the detailed level D_{d1} and D_{d2}, $project_{OBS'}(tclosure(D_{d1})) = project_{OBS'}(tclosure(D_{d2}))$, where $OBS' \subseteq OBS$ represents the available manifestations (parameter `Obs` of function `Abstract()`). Then, we prove that for any detailed diagnosis D_d, $project_{OBS'}(tclosure(D_d)) = project_{OBS'}(tclosure(D_a))$. Finally, we exploit this result to prove the theorem thesis.
In the following, $project_{OBS'}(tclosure(.))$ has been abbreviated in $tc_{OBS'}(.)$.

For step 1, we proceed by induction on the number of components which are assigned different behavioral modes in assignments D_{d1} and D_{d2}. The case $n = 1$ (i.e. $D_{d1} = \alpha \cup \{c_i(bm_r)\}$ and $D_{d2} = \alpha \cup \{c_i(bm_s)\}$) follows from the definition of indistinguishability of bm_r and bm_s.
For the inductive step, where $D_{d1} = \alpha 1 \cup \{c_i(bm_r)\}$, $D_{d2} = \alpha 2 \cup \{c_i(bm_s)\}$ and $\alpha 1, \alpha 2$ differ in assignments to n components, we note the following relations hold:
$tc_{OBS'}(\alpha 1 \cup c_i(bm_r)) = tc_{OBS'}(\alpha 1 \cup c_i(bm_s))$
from indistinguishability of bm_r and bm_s, and:
$tc_{OBS'}(\alpha 1 \cup c_i(bm_s)) = tc_{OBS'}(\alpha 2 \cup c_i(bm_s))$
from inductive hypothesis. It then follows that $tc_{OBS'}(\alpha 1 \cup c_i(bm_r)) = tc_{OBS'}(\alpha 2 \cup c_i(bm_s))$.

In order to carry step 2, we note that, since ν_i is substituted by `MergeModes()` wherever a mode bm_{ij_i} of its associated class π appears, the following holds:
$tc_{OBS'}(D_a) = \bigcup_{j_1,\ldots,j_n} tc_{OBS'}(\{c_1(bm_{1j_1}), \ldots, c_n(bm_{nj_n})\})$
But, in step 1, we have proved that all the terms of the union are equal. So, $tc_{OBS'}(D_a)$ is equal to the $tc_{OBS'}$ of any of the D_d.

We use this result for step 3: D_a is a diagnosis with the abstracted model iff $DT \cup \mathbf{CXT} \cup D_a \vdash \mathbf{OBS}'$; but, then, for any D_d the same entailment must hold, thus any D_d is a diagnosis at the detailed level. The converse is analogous. □

3.3 An Example

We end this section by illustrating how the abstraction algorithm works on a very simple SSD. Let the original Domain Theory DT contain the following clauses (figure 3 shows the associated System Structure Graph):

$s1(a) \wedge s2(a) \Rightarrow m1(x)$ $\quad$ $s1(a) \wedge s2(a) \Rightarrow m2(x)$
$s1(a) \wedge s2(b) \Rightarrow m1(x)$ $\quad$ $s1(a) \wedge s2(b) \Rightarrow m2(x)$
$s1(a) \wedge s2(c) \Rightarrow m1(x)$ $\quad$ $s1(a) \wedge s2(c) \Rightarrow m2(z)$
$s1(b) \wedge s2(a) \Rightarrow m1(y)$ $\quad$ $s1(b) \wedge s2(a) \Rightarrow m2(y)$
$s1(b) \wedge s2(b) \Rightarrow m1(y)$ $\quad$ $s1(b) \wedge s2(b) \Rightarrow m2(y)$
$s1(b) \wedge s2(c) \Rightarrow m1(y)$ $\quad$ $s1(b) \wedge s2(c) \Rightarrow m2(z)$

$i1(a) \wedge c1(a) \wedge c2(a) \Rightarrow s1(a)$ $\quad$ $i1(b) \wedge c1(a) \wedge c2(a) \Rightarrow s1(b)$
$i1(a) \wedge c1(a) \wedge c2(b) \Rightarrow s1(a)$ $\quad$ $i1(b) \wedge c1(a) \wedge c2(b) \Rightarrow s1(b)$
$i1(a) \wedge c1(a) \wedge c2(c) \Rightarrow s1(b)$ $\quad$ $i1(b) \wedge c1(a) \wedge c2(c) \Rightarrow s1(a)$
$i1(a) \wedge c1(b) \wedge c2(a) \Rightarrow s1(a)$ $\quad$ $i1(b) \wedge c1(b) \wedge c2(a) \Rightarrow s1(b)$
$i1(a) \wedge c1(b) \wedge c2(b) \Rightarrow s1(a)$ $\quad$ $i1(b) \wedge c1(b) \wedge c2(b) \Rightarrow s1(b)$
$i1(a) \wedge c1(b) \wedge c2(c) \Rightarrow s1(b)$ $\quad$ $i1(b) \wedge c1(b) \wedge c2(c) \Rightarrow s1(a)$

$i2(a) \wedge c3(a) \Rightarrow s2(c)$
$i2(a) \wedge c3(b) \Rightarrow s2(a)$
$i2(a) \wedge c3(c) \Rightarrow s2(b)$
$i2(b) \wedge c3(a) \Rightarrow s2(c)$
$i2(b) \wedge c3(b) \Rightarrow s2(a)$
$i2(b) \wedge c3(c) \Rightarrow s2(b)$

```
Function FindInfluences(N, ImmediateSuccessors)
     NodeInfluenceMatrix := ∅ × ∅ × ∅
     ForEach (bm_r ∈ modes(N), M ∈ ImmediateSuccessors, bm_s ∈ modes(M))
          Formulas := {clauses where N(bm_r) occurs in the body and M(bm_s) occurs in the head}
          α := false
          ForEach ((α_1 ∧ N(bm_r) ∧ α_2 ⇒ M(bm_s)) ∈ Formulas)
               α := α ∨ (α_1 ∧ α_2)
          Loop
          NodeInfluenceMatrix := NodeInfluenceMatrix ∪ {(N(bm_r), M(bm_s), normalize(α))}
     Loop
     Return NodeInfluenceMatrix
EndFunction

Function FindIndistinguishableModes(N, Modes, Nodes, InfluenceMatrix)
     M := first(Nodes)
     Π_cond := ∅
     ForEach (bm_r ∈ Modes)
          α := ⋃_{bm_s} ⟨ InfluenceMatrix(N(bm_r), M(bm_s)), M(bm_s) ⟩
          If (⟨α, π⟩ ∈ Π_cond) Then
               Π_cond := Π_cond − {⟨α, π⟩} ∪ {⟨α, π ∪ {bm_r}⟩}
          Else
               Π_cond := Π_cond ∪ {⟨α, {bm_r}⟩}
          EndIf
     Loop
     Π := ⋃_{⟨α,π⟩∈Π_cond} {π}
     If (tail(Nodes) ≠ ∅)
          ForEach (π ∈ Π)
               Π := Π − π ∪ FindIndistinguishableModes(N, π, tail(Nodes), InfluenceMatrix)
          Loop
     EndIf
     Return Π
EndFunction

Function MergeModes(N, Π, DT)
     DT' := ∅
     ForEach (π ∈ Π)
          ν := GenerateNewModeName(π)
          Formulas := {clauses for which ∃bm_r ∈ π s.t. N(bm_r) appears in the body or head}
          ForEach ((φ = α_1 ∧ N(bm_r) ∧ α_2 ⇒ M(bm_s)) ∈ Formulas)
               DT' := DT' ∪ {(α_1 ∧ N(ν) ∧ α_2 ⇒ M(bm_s))}
          Loop
          ForEach ((φ = α ⇒ N(bm_r)) ∈ Formulas)
               DT' := DT' ∪ {(α ⇒ N(ν))}
          Loop
     Loop
     Return DT'
EndFunction
```

Figure 2. Sketch of the main functions called by `Abstract()`

with $OBS = \{m1, m2\}$, $STATES = \{s1, s2\}$, $COMPS = \{c1, c2, c3\}$ and $CXT = \{i1, i2\}$.
Let the domains of the variables be as follows:
$DOM(m1) = \{x, y\}, DOM(m2) = \{x, y, z\}$
$DOM(s1) = \{a, b\}, DOM(s2) = \{a, b, c\}$
$DOM(c1) = \{a, b\}, DOM(c2) = DOM(c3) = \{a, b, c\}$
$DOM(i1) = \{a, b\}, DOM(i2) = \{a, b\}$
Furthermore, let's assume for simplicity that all the OBS are available at their maximum granularity.
The algorithm starts by trying to merge modes of $s1$. The `InfluenceMatrix()` entries relating $s1$ to $m1$ are:
$\langle s1(a), \{\langle m1(x), s2(a) \vee s2(b) \vee s2(c)\rangle, \langle m1(y), \bot\rangle\}\rangle$
$\langle s1(b), \{\langle m1(x), \bot\rangle, \langle m1(y), s2(a) \vee s2(b) \vee s2(c)\rangle\}\rangle$
it follows that modes a, b of $s1$ can't be merged. It is now $s2$ turn to be considered; the entries relating $s2$ to $m1$ are:
$\langle s2(a), \{\langle m1(x), s1(a)\rangle, \langle m1(y), s1(b)\rangle\}\rangle$
$\langle s2(b), \{\langle m1(x), s1(a)\rangle, \langle m1(y), s1(b)\rangle\}\rangle$
$\langle s2(c), \{\langle m1(x), s1(a)\rangle, \langle m1(y), s1(b)\rangle\}\rangle$
it may seem that modes a, b, c of $s2$ can be merged; however, $s2$ also influences another manifestation, $m2$:
$\langle s2(a), \{\langle m2(x), s1(a)\rangle, \langle m2(y), s1(b)\rangle, \langle m2(z), \bot\rangle\}\rangle$
$\langle s2(b), \{\langle m2(x), s1(a)\rangle, \langle m2(y), s1(b)\rangle, \langle m2(z), \bot\rangle\}\rangle$
$\langle s2(c), \{\langle m2(x), \bot\rangle, \langle m2(y), \bot\rangle, \langle m2(z), s1(a) \vee s1(b)\rangle\}\rangle$
we can thus merge modes a, b in new mode ab, but not mode c.
Having considered all the states, we now turn to the components, starting from $c1$:

$\langle c1(a), \{\langle s1(a), (i1(a) \wedge c2(a)) \vee (i1(a) \wedge c2(b)) \vee (i1(b) \wedge c2(c))\rangle,$
$\langle s1(b), (i1(a) \wedge c2(c)) \vee (i1(b) \wedge c2(a)) \vee (i1(b) \wedge c2(b))\rangle\}\rangle$
$\langle c1(b), \{\langle s1(a), (i1(a) \wedge c2(a)) \vee (i1(a) \wedge c2(b)) \vee (i1(b) \wedge c2(c))\rangle,$
$\langle s1(b), (i1(a) \wedge c2(c)) \vee (i1(b) \wedge c2(a)) \vee (i1(b) \wedge c2(b))\rangle\}\rangle$

modes a, b of $c1$ can then be merged in new mode ab; note that $c1$ goes into the trivial-nodes list, since all its domain has collapsed into a singleton. Similar arguments lead to merging modes a, b of $c2$; however, mode c of $c2$ can't be merged with the other two modes.
Component $c3$ is the only one left to be considered:
$\langle c3(a), \{\langle s2(ab), \bot\rangle, \langle s2(c), i2(a) \vee i2(b)\rangle\}\rangle$
$\langle c3(b), \{\langle s2(ab), i2(a) \vee i2(b)\rangle, \langle s2(c), \bot\rangle\}\rangle$
$\langle c3(c), \{\langle s2(ab), i2(a) \vee i2(b)\rangle, \langle s2(c), \bot\rangle\}\rangle$
we can merge modes b, c into a new node bc. Note that we can merge these modes only because we already unified modes a and b of $s2$; the importance of processing variables in the $\succ$ relation order is now evident.
Note also that we could have considered for abstraction $s2$ before $s1$, or $c2$ before $c1$ or after $c3$ since $s1, s2$ and $c1, c2, c3$ are not tied by the precedence order relation. It is easy to see that in such case the same mergings would have taken place anyway.

The output of the process described above results in a revised domain theory:

$s1(a) \wedge s2(ab) \Rightarrow m1(x)$
$s1(a) \wedge s2(c) \Rightarrow m1(x)$
$s1(b) \wedge s2(ab) \Rightarrow m1(y)$
$s1(b) \wedge s2(c) \Rightarrow m1(y)$

$s1(a) \wedge s2(ab) \Rightarrow m2(x)$
$s1(a) \wedge s2(c) \Rightarrow m2(z)$
$s1(b) \wedge s2(ab) \Rightarrow m2(y)$
$s1(b) \wedge s2(c) \Rightarrow m2(z)$

$i1(a) \wedge c1(ab) \wedge c2(ab) \Rightarrow s1(a)$
$i1(a) \wedge c1(ab) \wedge c2(c) \Rightarrow s1(b)$
$i1(b) \wedge c1(ab) \wedge c2(ab) \Rightarrow s1(b)$

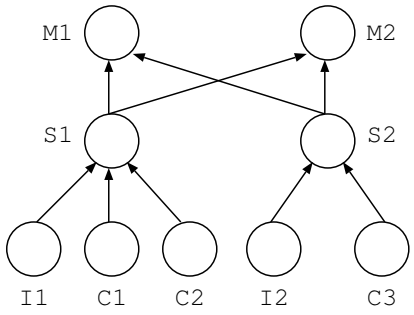

Figure 3. System Structure Graph for the Example Domain Theory

$i1(b) \wedge c1(ab) \wedge c2(c) \Rightarrow s1(a)$

$i2(a) \wedge c3(a) \Rightarrow s2(c)$
$i2(a) \wedge c3(bc) \Rightarrow s2(ab)$
$i2(b) \wedge c3(a) \Rightarrow s2(c)$
$i2(b) \wedge c3(bc) \Rightarrow s2(ab)$

and abstracted domains:
$DOM(m1) = \{x, y\}, DOM(m2) = \{x, y, z\}$
$DOM(s1) = \{a, b\}, DOM(s2) = \{ab, c\}$
$DOM(c1) = \{ab\}, DOM(c2) = \{ab, c\}, DOM(c3) = \{a, bc\}$
$DOM(i1) = \{a, b\}, DOM(i2) = \{a, b\}$

4 Using Abstract Models in On-Board Diagnosis

Having described how the abstraction algorithm works, we now consider how it can be used in real scenarios to practically improve the performance of the diagnostic problem solver.

A first scenario is when the manifestations of the system can be naturally subdivided in classes CL_k (see section 2); one such classes will contain all the manifestations (CL_{all}), another may contain only sensorized manifestations (CL_{sens}), further ones may exclude from CL_{sens} other groups of manifestations that can potentially all become unavailable together in some contexts. Similarly, manifestation granularities (expressed as abstraction functions τ_i) may be identified and associated to classes they apply to.
Equipped with this set of pairs $\langle CL_k, \tau_i\rangle$, we can generate off-line a corresponding set of models M_{ki}; when a specific diagnostic problem is presented to the on-line diagnostic agent, the minimal $\langle CL_k, \tau_i\rangle$ that covers the available observations is selected, and the corresponding model M_{ki} is used to compute diagnoses.

Sometimes, however, classes of manifestations (and their granularity) cannot be conveniently identified a-priori. In such cases we may want to compute an abstract model *on-demand*, given the particular CL_k and τ_i that have been identified as currently available [6].
The system may perform this on-line model synthesis as a lower priority task, asynchronously with the diagnostic tasks;

[6] How this info can be gathered, either manually or automatically, is out of the scope of the present paper

once the ad hoc M_{ki} has been computed it may be reused for many diagnostic problems until some conditions on the available observations or their granularity changes.
Obviously, time overhead is added by the computation of models but in some situations this may well be paid off by the benefits (see below). Moreover, experimental data presented below in section 5 show that such overhead may be in the order of the time needed for solving a few easy diagnostic cases (involving a single fault) or just a difficult one (involving multiple faults); keeping in mind that the abstraction algorithm is only run once whilst many diagnostic problems can exploit such a abstract model, this overhead may be acceptable.

In both the above scenarios, the number of returned diagnoses is reduced by returning diagnoses for the abstract model that correspond to sets of diagnoses for the detailed model that carry essentially the same information, as proved in section 3.2.
Moreover, whenever a diagnosis for the abstract model mentions a "compound mode" (i.e. a new mode introduced in place of a non-singleton set of indistinguishable modes), we explicitly know that the set of modes it represents couldn't be discriminated even in different contexts. Thus, in case further tests involving different contexts are planned, they shouldn't aim at that kind of discrimination.
Both reduced-size and increased informativity of the result should be helpful for the human or automatic supervisor which must interpret it and take action accordingly.

5 Experimental Results

We have implemented the algorithm described above as a module of the diagnostic agent for the space robotic arm SPIDER developed by ASI (Agenzia Spaziale Italiana); for a description of the diagnostic agent please see [12] and [11].
The model of the robotic arm (which obeys definition 2.1) is enough complex to represent an interesting test-bed: it consists of 35 assumables ($COMPS$) with an average 3.43 behavioral modes each, 45 manifestations (OBS) and 1143 formulas [7].

Observations in such a model are explicitly partitioned into two classes: sensorized (CL_{sens}) and non-sensorized (CL_{nosens}). While observations in CL_{sens} can reasonably be assumed to always be available, observations in CL_{nosens} are available through manual measurements that can be carried only during off-line diagnosis.

We have applied the abstraction algorithm to the model by passing CL_{sens} as the available observations (assuming $\tau_{OBS} = identity$, i.e. manifestations available at their maximum granularity) and obtained a simplified model as output. Table 1 shows some relevant static measures on the detailed and abstract models: the number of clauses has been reduced by 18.6%, and the average number of behavioral modes per system component has been reduced by 16.9%. Compilation of the detailed model in the abstract one took 1494msec of CPU time (all results in this section are referred to a Java implementation of both the diagnostic agent and the abstraction algorithm, compiled and run using jdk1.3 on a Sun Sparc Ultra 5 equipped with SunOS 5.8).

[7] The number of formulas is greatly reduced by the use of a noisy-max modeling technique, see [12]

model	clauses	modes avg
detailed	1143	3.43
abstract	930	2.85

Table 1. Comparison between abstract and detailed models

We have then compared the performance of the diagnostic agent when it uses the detailed (original) versus the generated abstract model. Using the simulator for the diagnostic agent, three test sets of 100 diagnostic problems each have been automatically generated; problems in test sets 1, 2 and 3 had 1, 2 and 3 faults injected respectively. Table 2 reports on the reduction of the average number of diagnoses returned. Particularly significant appear the reductions obtained in test set 2 (-43%) and test set 3 (-61.6%).
It should be noted that the diagnostic agent returns only preferred diagnoses (in particular, those that have a minimal number of faults), thus the reported reductions are obtained by compacting "good-quality" diagnoses, not by discarding implausible ones.

model	testset 1	testset 2	testset 3
detailed	5.0 ± 0.6	17.9 ± 3.6	123.3 ± 23.1
abstract	3.7 ± 0.4	10.2 ± 1.9	47.3 ± 8.4

Table 2. Average number of elementary diagnoses obtained with abstract and detailed models (confidence 95%)

Even if our diagnostic agent uses a compact encoding for candidate diagnoses during the search process, thus obtaining an optimized search space size that is not proportional to the number of diagnoses ([11]), the average time employed for solving problems using the abstract model appears to be at least no worse than that obtained by using the detailed model (see table 3).

model	testset 1	testset 2	testset 3
detailed	241 ± 19	337 ± 45	1212 ± 182
abstract	235 ± 25	259 ± 32	988 ± 153

Table 3. Average CPU times obtained with abstract and detailed models (in msec; confidence 95%)

Consistent results (both in terms of static reduction of the model size and reduction of diagnoses) have been obtained by applying the abstraction algorithm to other subclasses of manifestations in the model. Space precludes reporting them in this paper.

Please note that the faulty behavioral modes modeled for the components were the ones listed in the FMECA document for the real device, thus proving that the results obtained with the abstraction algorithm and reported above are of interest for a real-world system.

6 Related Work and Conclusions

Literature on MBD contains several proposals to use abstraction as a means of simplifying system model and, consequently, characterization and computation of diagnoses.

Some of them formulate *abstraction rules* and prove that abstractions obtained by their application preserve important properties, i.e. by reasoning at the abstract level we don't overlook any diagnoses ([9], [13]).

Among the rules proposed by Mozetič, rule 1 (Refinment/collapse of values) aims exactly at abstracting sets of values at the detailed level into a single value at a more abstract level. Compared to our approach, however, both Mozetič and Provan assume that the abstraction is done manually, by a human knowledgeable about the system behavior and structure. Moreover, they use the abstract models only in order to reduce the computational complexity, but still return detailed level diagnoses.

In a recent paper ([2]), the authors improve Mozetič approach so that the hierarchy (still manually provided) is automatically rearranged on a case by case basis in order not to hide any available observations from the abstract levels.

Sachenbacher and Struss ([14]) have defined a relational-based approach (i.e. the behavior model is given as a relation R among tuples of variables) for automated abstraction of variables domains and showed its usefulness in building system models by composing sub-system models and then abstracting away values details that are not of interest in the resulting model. In contrast to our approach, their work assumes that a desired abstraction τ_{targ} be given as part of the abstraction problem, together with the restrictions on available observations information τ_{obs} that appear also in our approach.

Travé-Massuyès, Escobet and Milne ([15]) define a notion of *indiscriminability* among faulted components which is somewhat similar to our notion of *indistinguishability* among behavioral modes. Their work, which is based on a relational model of the system, aims at suggesting the addition of sensors in order to make all the possible faults discriminable.

In this paper we have shown how abstraction of variable domains in propositional, qualitative system models, can significantly reduce the average number of admissible diagnoses when only a subset of observables is available.

This is particularly useful in on-line diagnosis, where limitations on the number and/or granularity of observables can likely apply.

The algorithm presented in the paper has a larger applicability than discussed so far. For example, it can be used as a support tool for diagnosability during system modeling (i.e. the algorithm can point out discrepancies between the granularity of the model being defined and that of the system observables).

There are many directions we are considering for extending our work. The current version of the algorithm assumes that in the device structure nodes are connected by at most one directed path, while representation of some systems of practical interest does not obey to this restriction.

We also could explore how our automatic abstraction techniques can be extended in order to merge together components whose contributions to the available observations are indiscriminable (i.e. introducing the notion of indistinguishable components).

REFERENCES

[1] F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano, and D. Theseider Dupré, 'Generating on-board diagnostics of dynamic automotive systems based on qualitative models', *AI Communications*, **12**, 33–43, (1999).

[2] L. Chittaro and R. Ranon, 'Hierarchical diagnosis guided by observations', in *Proc.* 17^{th} *Int. Joint Conference on Artificial Intelligence - IJCAI01*, pp. 573–578, Seattle, USA, (2001).

[3] L. Console and O. Dressler, 'Model-based diagnosis in the real world: lessons learned and challenges remaining', in *Proc.* 16^{th} *Int. Joint Conference on Artificial Intelligence - IJCAI99*, pp. 1393–1400, Stockholm, Sweden, (1999).

[4] L. Console and P. Torasso, 'A spectrum of logical definitions of model-based diagnosis', *Computational Intelligence*, **7**(3), 133–141, (1991).

[5] A. Darwiche, 'Model-based diagnosis using structured system descriptions', *Journal of Artificial Intelligence Research*, **8**, 165–222, (1998).

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

[7] J. de Kleer, O. Raiman, and M. Shirley, 'One step lookahead is pretty good', in *Proc.* 2^{nd} *Int. Workshop on Principles of Diagnosis - DX91*, pp. 136–142, Milan, Italy, (1991).

[8] W. Hamscher, 'Modeling digital circuits for troubleshooting', *Artificial Intelligence*, **51**, 223–271, (1991).

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

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

[11] L. Portinale and P. Torasso, 'Diagnosis as a variable assignment problem: a case study in a space robot fault diagnosis', in *Proc.* 16^{th} *Int. Joint Conference on Artificial Intelligence - IJCAI 99*, pp. 1087–1093, Stockholm, Sweden, (1999).

[12] L. Portinale, P. Torasso, and G. Correndo, 'Knowledge representation and reasoning for fault identification in a space robot arm', in *Proc.* 5^{th} *Int. Symposium on AI, Robotics and Automation is Space - iSAIRAS99*, pp. 539–546, Noordwijk, Holland, (1999).

[13] G. Provan, 'Hierarchical model-based diagnosis', in *Proc.* 12^{th} *Int. Workshop on Principles of Diagnosis - DX01*, pp. 329–362, San Sicario, Italy, (2001).

[14] M. Sachenbacher and P. Struss, 'Aqua: A framework for automated qualitative abstraction', in *Proc.* 15^{th} *Int. Workshop on Qualitative Reasoning - QR01*, San Antonio, USA, (2000).

[15] L. Travé-Massuyès, T. Escobet, and R. Milne, 'Model-based diagnosability and sensor placement. application to a frame 6 gas turbine sub-system', in *Proc.* 17^{th} *Int. Joint Conference on Artificial Intelligence - IJCAI01*, pp. 551–556, Seattle, USA, (2001).