Automatic Abstraction in Component-Based Diagnosis Driven by System Observability

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

Abstract

The paper addresses the problem of automatic abstraction of component variables in the context of Model Based Diagnosis, in order to produce models capable of deriving fewer and more general diagnoses when the current observability of the system is reduced. The notion of indiscriminability among faults of a set of components is introduced and constitutes the basis for a formal definition of admissible abstractions which preserve all the distinctions that are relevant for diagnosis given the current observability of the system. The automatic synthesis of abstract models further restricts abstractions such that the behavior of abstract components is expressed in terms of a simple and intuitive combination of the behavior of their subcomponents. As a validation of our proposal, we present experimental results which show the reduction in the number of diagnoses returned by a diagnostic agent for a space robotic arm.

1 Introduction

System model abstraction has been successfully exploited in many approaches to model-based diagnosis (MBD). The pioneer work of [Mozetič, 1991] and recent improvements proposed e.g. by [Provan, 2001] and [Chittaro and Ranon, 2001] mostly use abstraction in order to focus the diagnostic process and thus improve its efficiency. However, (flexible) abstraction has another main benefit, namely to provide fewer and more concise abstract diagnoses when it is not possible to discriminate among detailed diagnoses. The works by [Console and Theseider Dupré, 1994] and [Friedrich, 1993] accomplish this goal by including abstraction axioms in the Domain Theory and using preference criteria based on the abstraction level of diagnoses.

Recently, some authors have aimed at the same goal in a different way, namely automatic abstraction of the system model ([Sachenbacher and Struss, 2001], [Torasso and Torta, 2002]) [1]. If the available observables and/or their granularity are too coarse to distinguish among two or more behavioral modes of a component, or the distinction is not important for the considered system, a system model is automatically generated where such behavioral modes are merged into an abstracted behavioral mode. By using the abstracted model for diagnosis there's no loss of (important) information, while the number of returned diagnoses is reduced, and such diagnoses, by being "as abstract as possible", are more understandable for a human.

The work presented in this paper aims at extending previous works by introducing automatic abstraction of variables (i.e. components) in the presence of a reduced availability of the number and/or granularity of observables. Abstractions based on system observability are particularly relevant in the context of on-board diagnosis. Indeed, it is likely that when a system is operated on-board the only available measures are provided by sensors (which can themselves fail) and taking further measures manually is out of question. Moreover, on-board diagnosis is usually constrained by strict time and resources requirements: using an abstracted system model should yield savings in both the time and the space requirements of the diagnostic process.

Our proposal requires that abstractions do not cause any loss of diagnostic information (e.g. as in the incomplete abstractions discussed in [Autio and Reiter, 1998]) or loss of efficiency (e.g. due to increased fan-in as pointed out in [Provan, 2001]); moreover, we restrict the mapping from abstract components to their subcomponents to be enough simple and intuitive. In order to exclude all the undesired abstractions we introduce a precise definition of *admissible* abstraction, and further restrict the computation of abstractions through *cutoff* criteria which forbid admissible abstractions that may lead to computational inefficiencies.

As a running example to illustrate our definitions and algorithms, we will use throughout the paper the fragment of hydraulic circuit adapted from [Chittaro and Ranon, 2001] depicted in figure 1 **(a)**; table 1 reports its domain theory (valve modes *so* and *sc* abbreviate

[1] Previously an algorithm for automatic abstraction of components had been proposed in [Out *et al.*, 1994] for simpler models describing only normal behavior of the system

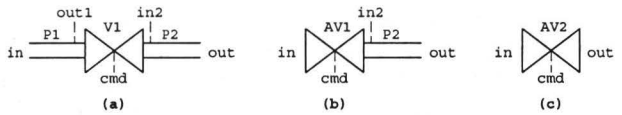

Figure 1: A fragment of an hydraulic circuit at three levels of abstraction

$valve(x) \wedge cmd_x(open) \wedge x(ok) \Rightarrow out_x = in_x$
$valve(x) \wedge cmd_x(open) \wedge x(leak) \Rightarrow out_x < in_x$
$valve(x) \wedge cmd_x(open) \wedge x(so) \Rightarrow out_x = in_x$
$valve(x) \wedge cmd_x(open) \wedge x(sc) \Rightarrow out_x = 0$
$valve(x) \wedge cmd_x(close) \wedge x(ok) \Rightarrow out_x = 0$
$valve(x) \wedge cmd_x(close) \wedge x(leak) \Rightarrow out_x = 0$
$valve(x) \wedge cmd_x(close) \wedge x(so) \Rightarrow out_x = in_x$
$valve(x) \wedge cmd_x(close) \wedge x(sc) \Rightarrow out_x = 0$
$pipe(x) \wedge x(ok) \Rightarrow out_x = in_x$
$pipe(x) \wedge x(br) \Rightarrow out_x = 0$

Table 1: Model of hydraulic components

stuck open and *stuck closed* respectively). Two possible abstractions of such system are shown in figures 1 **(b)** (where pipe $P1$ and valve $V1$ have been abstracted into valve $AV1$) and 1 **(c)** (where abstract valve $AV1$ and pipe $P2$ have been abstracted into valve $AV2$). As for experimental validation of our techniques, however, we will present results collected in a larger domain, namely the model of the SPIDER space robotic arm used in [Portinale and Torasso, 1999].

The paper is structured as follows. In section 2 we give a formal definition of what we consider an admissible abstract component. In section 3 we describe how the declarative notions introduced in 2 can be implemented computationally. In section 4 we present experimental results collected in the SPIDER robotic arm domain. Finally, in section 5 we compare our work to related papers and make some concluding remarks.

2 Abstractions Defined

We first report formal definitions of system model, diagnostic problem and diagnosis.

Definition 2.1 *A* System Description *(SD) is a 3-tuple* $\langle \mathcal{V}, DT, \mathcal{G} \rangle$ *where:*

- $\mathcal{V}$ *is a set of discrete variables partitioned in the following sorts: CXT (inputs), COMPS (components), STATES (endogenous variables), OBS (observables); DOM(v) is the finite domain of variable* $v \in \mathcal{V}$
- *DT (Domain Theory) is an acyclic set of Horn clauses defined over* $\mathcal{V}$ *representing the behavior of the system (under normal and abnormal conditions); we require that, given an instantiation of COMPS and CXT, the DT derives exactly one value for each other variable*
- $\mathcal{G}$ *(Causal Graph) is a DAG whose nodes are in* $\mathcal{V}$ *representing the causal structure of the system; 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*

It is worth noting that the fact that $\mathcal{G}$ is restricted to be a DAG does not mean that the system model is a tree in the sense e.g. of [Darwiche, 1998]; since we allow the existence of multiple directed paths between two nodes the associated jointree may well be cyclic. The DAG restriction just forbids feedback loops in the causal graph, which is a common assumption in approaches which deal with dynamic systems only if they are amenable to state-based diagnosis ([Struss, 1997]).

Example 2.1 Note that the System Description for the running sample system given in table 1 is expressed in a slightly different formalism than the one described in our definition, due to the use of equations and disequations on the right-hand side of formulas. However, it can be easily mapped to our formalism by modeling in_x and out_x as discrete $STATES$ variables by using qualitative deviations ([Struss *et al.*, 1996]); thus, for example, formula:

$valve(x) \wedge cmd_x(open) \wedge x(ok) \Rightarrow out_x = in_x$

is mapped to a set of three formulas:

$in_x(\langle +, 0 \rangle) \wedge cmd_x(open) \wedge x(ok) \Rightarrow out_x(\langle +, 0 \rangle)$
$in_x(\langle +, - \rangle) \wedge cmd_x(open) \wedge x(ok) \Rightarrow out_x(\langle +, - \rangle)$
$in_x(\langle 0, - \rangle) \wedge cmd_x(open) \wedge x(ok) \Rightarrow out_x(\langle 0, - \rangle)$

where $v(\langle [val], [\Delta val] \rangle)$ means that variable v has a value with sign $[val]$ and its deviation from the nominal value has sign $[\Delta val]$. In the rest of the examples we continue to use the notation of [Chittaro and Ranon, 2001] simply because it is more compact. □

Definition 2.2 *A diagnostic problem is a 3-tuple* $DP = \langle SD, \mathbf{OBS}, \mathbf{CXT} \rangle$ *where SD is the System Description,* **OBS** *is an instantiation of OBS and* **CXT** *is an instantiation of CXT*

Definition 2.3 *Given a diagnostic problem* $DP = \langle SD, \mathbf{OBS}, \mathbf{CXT} \rangle$ *an instantiation* $D = \{C_1(bm_1), \ldots, C_n(bm_n)\}$ *of COMPS is a diagnosis for DP iff:*

$\forall M(val) \in \mathbf{OBS} \quad DT \cup \mathbf{CXT} \cup D \vdash M(val)$ [2]

As noted in the introduction, there are real scenarios where not all the observables OBS are available and/or their granularity is reduced. We identify the available observables with a set $OBS_{AV} \subseteq OBS$; we also assume that a *granularity mapping* Π is given s.t. $\Pi(M(val))$ maps an instantiation of $M \in OBS_{AV}$ to a possibly more abstract instantiation $M(aval)$. If $\forall M \in OBS_{AV}, \forall val \in DOM(M) : \Pi(M(val)) = M(val)$ then there is no loss of granularity at all; in this particular case we denote Π with Π_{id}. For example, in figure 1 **(a)**, $OBS_{AV} = OBS = \{out1, in2, out\}$; in 1 **(b)**, $OBS_{AV} = \{in2, out\}$ and in 1 **(c)**, $OBS_{AV} = \{out\}$; as for Π we may consider a situation where $out_x = in_x$ and $out_x < in_x$ have been mapped to a coarser value $out_x \neq 0$. The important point about OBS_{AV} and Π is that these reductions of the observability of the system can cause the model to become less discriminant

[2] This definition of diagnosis is fully abductive; however results presented in this paper apply equally well to consistency-based diagnosis

and thus different faults of the same component and/or of different components to become indiscriminable. The following definition introduces the notion of indiscriminability among instantiations of subsets of $COMPS$.

Definition 2.4 *Let $SCOMPS$ be a subset of $COMPS$, OBS_{AV} the set of available observables and Π a granularity mapping. We say that two instantiations* **SCOMPS1, SCOMPS2** *of $SCOMPS$ are* $\langle OBS_{AV}, \Pi\rangle$-indiscriminable *iff for any instantiation* **CXT** *of CXT and any instantiation* **OTHERS** *of $COMPS \backslash SCOMPS$ the following holds* [3]*:*

$$\Pi(tclosure_{OBS_{AV}}(\mathbf{OTHERS} \cup \mathbf{SCOMPS1})) = \Pi(tclosure_{OBS_{AV}}(\mathbf{OTHERS} \cup \mathbf{SCOMPS2}))$$

Note that the $\langle OBS_{AV}, \Pi\rangle$-indiscriminability relation induces a partition into $\langle OBS_{AV}, \Pi\rangle$-indiscriminability classes of the set of possible instantiations of $SCOMPS$. Also note that, when $|SCOMPS| = 1$, indiscriminability coincides with the indistinguishability among behavioral modes introduced in [Torasso and Torta, 2002].

Example 2.2 In the abstraction example of figure 1 **(b)**, under context $cmd = open$ indiscriminable instances of $P1, V1$ are grouped in the following sets: $C1 = \{\langle ok, ok\rangle, \langle ok, so\rangle\}$ (which derive $in2 = in$); $C2 = \{\langle ok, leak\rangle\}$ (which derives $in2 < in$); $C3 = \{\langle ok, sc\rangle, \langle br, ok\rangle, \langle br, leak\rangle, \langle br, so\rangle, \langle br, sc\rangle\}$ (which derive $in2 = 0$). By also considering context $cmd = close$, set $C1$ is further split in two distinct sets: $C11 = \{\langle ok, ok\rangle\}$ (which derives $in2 = 0$) and $C12 = \{\langle ok, so\rangle\}$ (which derives $in2 = in$). If in a specific diagnostic problem we observe, for instance, $in2 = 0$ given $cmd = open$, the set of diagnoses is given by $C3$ [4]. □

For now, just note how this notion of indiscriminability is the basis for any potential abstraction. If, indeed, two instantiations **SCOMPS1** and **SCOMPS2** of $SCOMPS \subseteq COMPS$ are indiscriminable, this means that whenever **SCOMPS1** $\cup$ **OTHERS** is a diagnosis for a given DP, **SCOMPS2** $\cup$ **OTHERS** is another, indiscriminable, diagnosis for DP.

We now introduce a weak notion of abstraction where, as it is common in structural abstractions, abstract components are recursively built bottom-up starting with the primitive components.

Definition 2.5 *Given a set $COMPS = \{C_1, \ldots, C_n\}$ of component variables, a* components abstraction mapping $\mathcal{AM}$ *of $COMPS$ defines a set $COMPS_A = \{AC_1, \ldots, AC_m\}$ of discrete variables (*abstract components*) and associates to each $AC_i \in COMPS_A$ one or more $C_j \in COMPS$ (*subcomponents *of AC_i) s.t. each component in $COMPS$ is the subcomponent of exactly one abstract component. Moreover, $\mathcal{AM}$ associates, to each abstract component AC a* definition def_{AC}*, which is a characterization of the behavioral modes of AC in terms of the behavioral modes of its subcomponents. More precisely, an abstract component and its definition are built hierarchically as follows:*

- *if $C \in COMPS$, AC is a* simple *abstract component if its definition def_{AC} associates to each $abm \in DOM(AC)$ a formula $def_{abm} = C(bm_1) \vee \ldots \vee C(bm_k)$ s.t. $bm_1, \ldots, bm_k \in DOM(C)$* [5]*; in the* trivial *case, AC has the same domain as C and $\forall bm \in DOM(C) : def_{bm} = C(bm)$*
- *if AC', AC'' are abstract components with disjoint sets of subcomponents $SCOMPS', SCOMPS''$ then AC is an abstract component with subcomponents $SCOMPS' \cup SCOMPS''$ if def_{AC} associates to each $abm \in DOM(AC)$ a definition def_{abm} which is a logical formula built by connecting definitions $def_{bm'}, bm' \in DOM(AC')$ with definitions $def_{bm''}, bm'' \in DOM(AC'')$ through $\vee, \wedge$ and $\sim$*

The definition def_{AC} of AC thus specifies a relation between instantiations of the subcomponents of AC and instantiations (i.e. behavioral modes) of AC itself. However we need to put some restrictions on these relations in order to match our intuitions about what is an "admissible" abstraction.

Definition 2.6 *Given a simple abstract component the definitions associated to its behavioral modes are said to be* admissible. *For a non-simple abstract component AC which is the composition of abstract components AC', AC'', an* admissible definition *for $abm \in DOM(AC)$ is defined as follows:*

1. conjunction*: $def_{abm} = def_{abm'} \wedge def_{abm''}, abm' \in DOM(AC'), abm'' \in DOM(AC'')$ is admissible*
2. generalized OR*: if $def'_1, \ldots, def'_l \in DOM(AC')$ and $def''_1, \ldots, def''_m \in DOM(AC'')$ then:*
   $def_{abm} = (def'_1 \wedge \sim def''_2 \wedge \ldots \wedge \sim def''_m) \vee (def''_1 \wedge \sim def'_2 \wedge \ldots \wedge \sim def'_l)$
   is admissible. As an important special case, when $l = m = 1$, def_{abm} is a canonical OR

Admissible definitions capture common abstractions, such as the case where the abstract component is OK if all its subcomponents are OK (conjunction) and faulty if at least one of its subcomponents is faulty (canonical OR). Moreover, since we address the case of components with multiple behavioral modes, we extend canonical OR with exceptions (generalized OR); clearly, the maximum number of exceptions allowed in a generalized OR should be a configurable parameter of the abstraction algorithm.

It is worth noting that the proposed operators, although chosen according to the rationale just exposed, are meant by no means to be the only possible choice in order to

[3] Given an instantiation **COMPS** of $COMPS$ we denote with $tclosure_{OBS_{AV}}(\mathbf{COMPS})$ the set of instantiations of variables in OBS_{AV} derived from $(DT \cup \mathbf{CXT} \cup \mathbf{COMPS})$

[4] There are $|C3| = 5$ indiscriminable diagnoses; 2 of them, namely $\langle ok, sc\rangle$ and $\langle br, ok\rangle$, can be regarded as preferred diagnoses since they involve only one fault

[5] This case has the purpose of including behavioral modes abstraction as described in [Torasso and Torta, 2002] in our framework: $bm_1, \ldots, bm_k$ are abstracted in the single behavioral mode abm

make our approach to abstraction applicable; if more operators or a different set of operators would fit better particular systems or families of systems, new operators could simply be plugged-in and replace the ones we have defined.

Example 2.3 In the abstraction example of figure 1 **(b)**, the behavioral modes of the abstract component $AV1$ can be expressed as admissible definitions over primitive components $P1, V1$:

$def_{abm1} = P1(ok) \wedge V1(ok)$ (conjunction)
$def_{abm2} = P1(ok) \wedge V1(leak)$ (conjunction)
$def_{abm3} = P1(ok) \wedge V1(so)$ (conjunction)
$def_{abm4} = P1(br) \vee V1(sc)$ (canonical OR) □

Armed with the *admissible definitions* for behavioral modes we can now formally identify the abstraction mappings we are interested in.

Definition 2.7 *Given a system model SD, a components abstraction mapping $\mathcal{AM}$ of COMPS, a set $OBS_{AV} \subseteq OBS$ and a granularity mapping Π, we say that $\mathcal{AM}$ is* admissible *w.r.t. SD, OBS_{AV}, Π iff for each abstract component AC with subcomponents SCOMPS:*

1. admissible behavioral modes*: each $def_k \in def_{AC}$ is admissible in the sense of definition 2.6*
2. mutual exclusion*: for any two distinct $def_k, def_l \in def_{AC}$, and any instantiation* **COMPS** *of COMPS:* $\mathbf{COMPS} \cup \{def_k \wedge def_l\} \vdash \perp$
3. completeness*: for any instantiation* **COMPS** *of COMPS:* $\mathbf{COMPS} \vdash \bigvee_{k=1,\ldots,|DOM(AC)|} def_k$
4. correctness*: given $def_k \in def_{AC}$, the set of instantiations of SCOMPS which satisfy def_k is a $\langle OBS_{AV}, \Pi\rangle$-indiscriminability class* [6]

Example 2.4 The abstraction example of figure 1 **(b)**, is admissible w.r.t. $OBS_{AV} = \{in2, out\}$ and $\Pi \equiv \Pi_{id}$, if the behavioral modes definitions of example 2.3 are used. Such definitions are admissible, moreover there is a 1 : 1 mapping between them and the $\langle OBS_{AV}, \Pi\rangle$-indiscriminability classes shown in example 2.2; in particular def_{abm1} corresponds to $C11$, def_{abm2} to $C2$, def_{abm3} to $C12$ and def_{abm4} to $C3$. It follows that the mutual exclusion, completeness and correctness conditions are also satisfied □

Note that given an admissible components abstraction mapping $\mathcal{AM}$, to each instantiation **COMPS** of $COMPS$ corresponds exactly one instantiation $\mathbf{COMPS_A}$ of $COMPS_A$ consistent with $COMPS$ given the definitions of elements in $COMPS_A$. We say that $\mathbf{COMPS_A}$ is the *abstraction* of **COMPS** according to $\mathcal{AM}$.

3 Computing Abstractions

The hierarchical way abstract components are defined in section 2 suggests that, after an initial step of behavioral modes abstraction, the computational process can produce new abstract components incrementally, by merging only two components at each iteration. After some finite number of iterations, arbitrarily complex abstract components can be produced.

As already mentioned in section 1, however, the admissibility of a component abstraction is not enough in order to produce useful and meaningful abstractions. We thus introduce some *cutoff* criteria on abstractions over two components (i.e. single iterations), to be enforced by the computational process.

First, we don't want to have a different behavioral mode of the abstract component for each combination of the behavioral modes of its subcomponents (limited-domain criterion); a proliferation of behavioral modes in the abstract component has negative effects on both the efficiency of diagnosis and the understandability of abstract diagnoses. We chose to impose the not too-restrictive limit $|DOM(AC)| \leq |DOM(AC')| + |DOM(AC'')|$. Second, we must be able to control the fan-in of the abstract components; indeed, if a structure-based diagnostic algorithm is used ([Darwiche, 1998]), introducing an abstract component that has a fan-in (much) larger than that of all its subcomponents leads to computational inefficiencies as pointed out in [Provan, 2001]. The limit imposed on the fan-in of the abstract component (fan-in criterion) can vary from the maximum among the fan-ins of its subcomponents to the sum of such fan-ins; the choice should be driven by the type of diagnostic algorithm to be used with the abstracted model as well as by specific characteristics of the model under consideration. In the experiments reported in section 4 we have obtained significant results by restricting the fan-in of abstract components to be at most the maximum fan-in among their subcomponents.

The following is an high-level description of the computational process:

```
Function Abstract(SD, OBS_AV, Π)
  SD := BMAbstract(SD, OBS_AV, Π)
  While (Oracle(SD) = ⟨C_i, C_j⟩)
    LocalDT = CutDT(DT, C_i, C_j)
    If (MergeComps(LocalDT, Ci, Cj) = ⟨ AC, defAC ⟩)
      Comps = Comps \ {Ci, Cj} ∪ {AC}
      DT = ReviseDT(⟨ AC, defAC ⟩, Ci, Cj)
      G = RecomputeGraph(DT)
  Return SD
EndFunction
```

The invocation of function *BMAbstract()* builds simple abstract components by abstracting behavioral modes that result indistinguishable given the current observability expressed by OBS_{AV}, Π; then a while loop is entered.

Function *Oracle()* selects the next two candidates C_i, C_j for abstraction according to the strategies outlined in the next paragraph. Then $LocalDT$, i.e. the portion of DT relevant for computing the influence of C_i, C_j over the values of observables in OBS_{AV}, is isolated by calling $CutDT()$. The causal graph $\mathcal{G}_{loc}$ associated to $LocalDT$ essentially contains the paths from C_i, C_j to nodes in OBS_{AV}; all the parents of the nodes on such paths are

[6] Note that this guarantees that the behavioral modes of the abstract component are all distinguishable in the sense of [Torasso and Torta, 2002]

also added. It is easy to see that, given the nodes in $\mathcal{G}_{loc}$, the remaining nodes in the original $\mathcal{G}$ do not play any role in the way C_i, C_j influence the variables in OBS_{AV}. Function *MergeComps()* (see paragraph 3.2) then tries to compute the definition of the new abstract component given *LocalDT* and in case it succeeds, *SD* is updated accordingly by function *ReviseDT*(). Intuitively, the formulas in *LocalDT* are replaced with a set of formulas whose antecedents mention the new abstract component *AC*. Given the antecedent of a formula mentioning *AC* the consequent is computed according to the definition def_{AC}. For example, considering again figure 1 **(b)**, we introduce the formula $(cmd = open) \wedge AV1(abm1) \Rightarrow (in2 = in)$ since $(cmd = open) \wedge def_{abm1} \vdash (in2 = in)$ (recall that $def_{abm1} = P1(ok) \wedge V1(ok)$); similarly, we introduce formulas $(cmd = open) \wedge AV1(abm2) \Rightarrow (in2 < in)$, $(cmd = open) \wedge AV1(abm3) \Rightarrow (in2 = in)$, etc. [7]. The whole process terminates when function *Oracle()* has no new suggestions for further abstractions.

3.1 The Oracle

Function *Oracle*() must choose, at each iteration, a pair of candidate components for abstraction. Since a search over the entire space of potential abstractions would be prohibitive, the function is based on a greedy heuristic that tries to achieve a good overall abstraction without doing any backtracking. It is worth noting that the greedy approach reduces the worst-case search performed by *Oracle*() to the evaluation of all the possible pairs of components before returning the chosen pair; this can happen at most $|COMPS|$ times since every time an abstraction takes place the number of components is reduced by one and thus the search space size is clearly polynomial in $|COMPS|$.

In our experience it turned out that the heuristic H_A based on the following two principles could achieve significant results. First, H_A follows a *locality principle* for choosing C_i, C_j. Manually written structural abstractions, indeed, tend to consist in hierarchies such that at each level of the hierarchy the new abstractions involve components that are structurally close to one another. This usually has the advantage of building abstract components that have a fan-in comparable to that of their subcomponents and have a limited and meaningful set of behavioral modes. Two good examples of structural patterns that follow this principle are components connected in sequence and in parallel [8].
Second, the heuristic prefers pairs of components that are connected to the same observables in OBS_{AV}; this follows from the fact that if at least one of the two components is observable separately from the other, it is more unlikely to find indiscriminable instantiations of the two components.

[7] It is easy to see that, by interpreting *abm*1, *abm*2,abm3, *abm*4 as *ok*, *leak*, *so*, *sc* respectively, *AV*1 behavior is exactly that of a valve

[8] Note that the structural notions of vicinity, sequentiality and parallelism can be naturally transposed in terms of relationships in the causal graph $\mathcal{G}$

While evaluating a pair of candidates, *Oracle*() immediately enforces the fan-in criterion (this check can be easily performed at this stage). As for the limited-domain cutoff criterion and the actual feasibility of an admissible abstraction, *Oracle*() just "trusts" the heuristic H_A, and defers to *MergeComps*() the actual enforcement. There's thus no warranty that the components selected by *Oracle*() will end up being merged but just a good chance of it to happen.

In the current implementation, *Oracle*() terminates when it can't find any pair of components which meet the limited fan-in criterion, are connected in sequence or parallel and influence the same set of observables.

Example 3.1 Given the system in figure 1 **(a)** and $OBS_{AV} = \{in2, out\}$, *Oracle*() selects $P1, V1$ as candidates according to heuristic H_A: $P1, P2$, e.g., are further away and there's also an observable point between them, namely *in*2. Since the inputs to the potential abstract component *AV*1 would be *in*, *cmd* and the inputs to *V*1 are *out*1, *cmd* there is no increase in the fan-in of the abstract component. Thus $P1, V1$ are returned □

3.2 Abstraction of Two Components

Once two candidate components C_i, C_j have been selected, function *MergeComps*() tries to merge them into a single abstract component. The following is a sketch of the function:

```
Function MergeComps(LocalDT, C_i, C_j)
  P = FindIndiscriminable(LocalDT, C_i, C_j)
  If (|P| > |DOM(C_i)| + |DOM(C_j)|) Then Return NULL
  AC = NewCompName(); def_AC = ∅
  ForEach p ∈ P
    abm = NewBMName()
    def_abm = MakeABMDefinition(p)
    If (def_abm = NULL) Then Return NULL
    Else def_AC = def_AC ∪ {(abm, def_abm)}
  Return ⟨AC, def_AC⟩
EndFunction
```

First, the set $DOM(C_i) \times DOM(C_j)$ is partioned into indiscriminability classes by function *FindIndiscriminable*(). Such function considers in turn each observable M reachable from C_i, C_j and computes the set $SN(M)$ of source nodes (i.e. nodes without parents) in $\mathcal{G}_{loc}$ connected to M (excluding C_i, C_j); instantiations of the source nodes represent the contexts under which C_i, C_j influence the value of M. For each instantiation of $SN(M)$ then *FindIndiscriminable*() computes the transitive closure of each pair of behavioral modes in $DOM(C_i) \times DOM(C_j)$ and gradually refines the partition by putting into separate classes pairs that cause different values for M. It is easy to see that after *FindIndiscriminable*() has looped over each $M \in OBS_{AV}$ and each instantiation of $SN(M)$, the resulting partition $\mathcal{P}$ of $DOM(C_i) \times DOM(C_j)$ consists exactly in the indiscriminability classes of definition 2.4. Indiscriminability classes of $\mathcal{P}$ form the basis for building abstract behavioral modes definitions: associating exactly one abstract behavioral mode to each of them, guarantees that the mutual exclusion, completeness and

correctness conditions given in definition 2.7 are automatically satisfied. Since the number of classes in the partition $\mathcal{P}$ corresponds to the number of abstract behavioral modes to be eventually generated, the limited-domain cutoff criterion is applied to $|\mathcal{P}|$.
If the check is passed successfully, the generation of the definitions for the abstract behavioral modes starts, by considering an indiscriminability class p at a time and calling function $MakeABMDefinition()$. Such function tries to build an admissible behavioral mode definition by considering the admissible forms in the same order as given in definition 2.6; if it does not succeed, it returns $NULL$ and the abstraction of C_i, C_j fails.

Example 3.2 Given that $Oracle()$ has selected $P1, V1$ as candidates for abstraction, the admissible abstraction described in example 2.4 is computed by $MergeComps()$ in the following way: function $FindIndiscriminable()$ computes a partition $\mathcal{P}$ whose indiscriminability classes are the ones mentioned in example 2.2; since $|\mathcal{P}| = 4 < 6 = |DOM(P1)| + |DOM(V1)|$ the cutoff check is passed; then, function $MakeABMDefinition()$ builds, for each indiscriminability class, the corresponding admissible definition shown in example 2.3 □

3.3 Correctness

We now state two correctness results concerning algorithm *Abstract*(). Their validity follows intuitively by the description of the algorithm given in the previous paragraphs; because of lack of space we omit the formal proofs. The first property just states that the algorithm builds abstractions according to definitions in section 2.

Property 3.1 *Function Abstract() builds an admissible components abstraction mapping*

The second property makes explicit the correspondence between detailed and abstract diagnoses.

Property 3.2 *Let SD_A be the system description obtained from SD, OBS_{AV}, Π by applying Abstract() and $\mathcal{AM}$ the associated admissible abstraction mapping. Given a diagnostic problem $DP = \langle SD, \mathbf{OBS}, \mathbf{CXT} \rangle$ and the corresponding abstract diagnostic problem $DP_A = \langle SD_A, \mathbf{OBS_{AV}}, \mathbf{CXT} \rangle$, D is a diagnosis for DP iff its abstraction D_A according to $\mathcal{AM}$ is a diagnosis for DP_A*

4 Experimental Results

For testing the approach described above we have used the model of the space robotic arm SPIDER ([Portinale and Torasso, 1999]), consisting in 35 components with an average 3.43 behavioral modes each, 45 observables and 1143 formulas. All the tests have been performed using a Java implementation running on a Sun Sparc Ultra 5 equipped with SunOS 5.8.
Please note that the SPIDER system model expresses causal relationships among the variables: for example, a failure in an electrical component can influence the temperature measured at some other physically neighboring components.

model	testset 1	testset 2	testset 3
detailed	5.1 ± 0.3	22.0 ± 2.5	123.3 ± 23.1
abstract1	3.7 ± 0.2	11.7 ± 1.3	47.4 ± 8.4
abstract2	1.6 ± 0.1	2.1 ± 0.2	3.5 ± 0.5

Table 2: Average number of preferred diag. (conf. 95%)

We have applied the *Abstract*() algorithm by considering as available observables the set of 29 sensorized observables at their maximum granularity. The resulting model, computed in about 1sec, was reduced to have 21 components with an average 2.76 behavioral modes and 548 formulas. Table 2 reports the average number of preferred (i.e. minimum fault cardinality) diagnoses produced by the diagnostic agent when using respectively the original model (*detailed*), the model obtained by performing only behavioral modes abstraction (*abstract*1) and the model created by *Abstract*() (*abstract*2). The three columns report results for three testsets of 250 cases each. The difference between the testsets lies in the number of faults (1, 2 and 3 respectively) injected in each test case.
The average times employed to solve a test case using the *detailed* model were, with a 95% confidence, 72 ± 4 msec (*testset*1), 135 ± 9 msec (*testset*2) and 333 ± 54 msec (*testset*3); by using the *abstract*2 model the average times dropped to 35 ± 3 msec (*testset*1), 41 ± 3 msec (*testset*2) and 45 ± 8 msec (*testset*3).

5 Related Work and Conclusions

As noted in the introduction only a few methods have been proposed for automatic model abstraction in the context of MBD. In particular, [Sachenbacher and Struss, 2001] and [Torasso and Torta, 2002] aim at the abstraction of the values of variables in the model while the present work aims also at automatically abstracting component variables. In [Out *et al.*, 1994] the authors introduce the notion of *ID-hierarchies* of abstract components which, as our admissible abstraction mappings, preserve a strict correspondence between an abstract diagnosis and the set of detailed diagnoses consistent with it. However, their work deals with models which represent only normal behavior of the system, which results in a significant simplification of the synthesis of behavioral modes definitions for the abstract components. In particular, the partition of the instantiations of the subcomponents is always reduced to two indiscriminability classes: a singleton *OK-class* (containing the unique instance where all subcomponents are *OK*) and an *AB-class* (containing all the remaining instances, where at least one subcomponent is *AB*). Since there's no fault model, the only instance which predicts values for the observables is the one in the OK-class; instances in the AB-class do not predict anything about observable values, and thus can be safely put in the same indiscriminability class without computing any transitive closure.

Among the methods based on manual abstraction,

[Friedrich, 1993] proposes a notion of diagnosis (theory diagnoses) which exploits *ab*-clauses explicitly added to the domain theory in order to compute abstract diagnoses. Similarly, in [Console and Theseider Dupré, 1994] the model is augmented with abstraction axioms which model *is-a* relationships; in such model also observables can be expressed at different levels of abstraction. In both works the level of abstraction of diagnoses is flexible (i.e. a diagnosis can mix elements at different levels of abstraction) and is driven by the specific diagnostic case at hand (i.e. values of the observables and contexts). Our abstraction algorithm automatically synthesizes a single abstract level given the knowledge of which variables are observable at which granularity; we assume that the changes in the availability of observables are rare, so that the produced models are reused for many diagnostic cases. Moreover, the kinds of relationships that we allow among abstract component behavior and its subcomponents behavior are less restrictive than *ab*-clauses and *is-a* relationships.

The definition of admissible components abstractions given in this paper aims at building abstract models that can completely replace the detailed models without any loss of discriminability. This is different from the goal of [Mozetič, 1991] and its improvements, where the abstract model is viewed as a focusing mechanism, and is used in conjunction with the detailed model in order to improve efficiency. Our notion of indiscriminability enforces abstractions where a diagnosis D_a at the abstract level corresponds exactly to the set of detailed diagnoses whose abstraction is D_a (property 3.2). As shown by [Autio and Reiter, 1998] this is not the case for some common abstractions: in the example of the abstract NOR gate consisting of an OR and a NOT gates, the *ab*-clause $AB(NOR) \equiv AB(OR) \vee AB(NOT)$ does not correspond to an indiscriminability class, since the case where both OR and NOT are faulty has a different behavior than the cases where only one fault is present. Our algorithm does not synthesize such *ab*-clause, thus avoiding the consequent loss of diagnostic information.

Most of the computational effort of our approach is devoted to the computation of the indiscriminability classes; once they have been identified, the generation of behavioral modes definitions from each of them is cheap and straightforward. An alternative approach that could be worth exploring is constructive inductive learning, where techniques have been developed for the automatic synthesis of complex attributes (see e.g. [Pagallo and Haussler, 1990]).

Another interesting point to explore would be to study more sophisticated versions of the *Oracle*() function, able to recognize a wider variety of patterns in the causal graph and possibly to look ahead one or more steps in order to select the candidate pair of components; in order to keep the complexity of the search manageable we believe that such extended versions of *Oracle*() should be supported by additional domain-specific knowledge (for instance, whether certain kinds of components are likely to be connected in ways that form admissible abstract components).

In conclusion, performing components abstraction led to dramatic reductions in the number of returned diagnoses in the experimental domain, outperforming the application of behavioral modes abstraction alone (see section 4). The reduction in the model size also resulted in significant time savings for the diagnostic process.

References

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

[Chittaro and Ranon, 2001] L. Chittaro and R. Ranon. Hierarchical diagnosis guided by observations. In *Proc. IJCAI01*, pages 573–578, 2001.

[Console and Theseider Dupré, 1994] L. Console and D. Theseider Dupré. Abductive reasoning with abstraction axioms. *LNCS*, 810:98–112, 1994.

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

[Friedrich, 1993] G. Friedrich. Theory diagnoses: A concise characterization of faulty systems. In *Proc. IJCAI93*, pages 1466–1471, 1993.

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

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

[Pagallo and Haussler, 1990] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

[Portinale and Torasso, 1999] L. Portinale and P. Torasso. Diagnosis as a variable assignment problem: a case study in a space robot fault diagnosis. In *Proc. IJCAI99*, pages 1087–1093, 1999.

[Provan, 2001] G. Provan. Hierarchical model-based diagnosis. In *Proc. DX01*, pages 329–362, 2001.

[Sachenbacher and Struss, 2001] M. Sachenbacher and P. Struss. Aqua: A framework for automated qualitative abstraction. In *Proc. QR01*, 2001.

[Struss *et al.*, 1996] P. Struss, A. Malik, and M. Sachenbacher. Qualitative modeling is the key to automated diagnosis. In *Proc. IFAC96*, 1996.

[Struss, 1997] P. Struss. Fundamentals of model-based diagnosis of dynamic systems. In *Proc. IJCAI97*, 1997.

[Torasso and Torta, 2002] P. Torasso and G. Torta. Merging indiscriminable diagnoses: an approach based on automatic domains abstraction. In *Proc. DX02*, pages 43–50, 2002.