Annals of Mathematics and Artificial Intelligence 11(1994)283–296

On the construction of hierarchic models*

D-J. Out, R.P. van Rikxoort and R.R. Bakker
University of Twente, Department of Computer Science, P.O. Box 217, 7500 AE Enschede, The Netherlands

One of the main problems in the field of model-based diagnosis of technical systems today is finding the most useful model or models of the system being diagnosed. Often, a model showing the physical components and the connections between them is all that is available. As systems grow larger and larger, the run-time performance of diagnostic algorithms decreases considerably when using these detailed models. A solution to this problem is using a hierarchic model. This allows us to first diagnose the system using an abstract model, and then use this solution to guide the diagnostic process using a more detailed model. The main problem with this approach is acquiring the hierarchic model. We give a generic hierarchic diagnostic algorithm and show how the use of certain classes of hierarchic models can increase the performance of this algorithm. We then present linear time algorithms for the automatic construction of these hierarchic models, using the detailed model and extra information about cost of probing points and invertibility of components.

1. Introduction

Model-based diagnosis of technical systems [2, 9] is an approach that uses a model of a system for simulation and reasoning. The model is used to detect discrepancies between observed behavior and model-computed behavior. These discrepancies form the basis for the derivation of the diagnoses of the system.

A model of a technical system consists of (1) components and connections between these components, and (2) behavioral descriptions of these components. If a component is working its behavior is equal to its behavioral description, if it is not working it can exhibit any behavior[1]. In model-based diagnosis we try to find sets of components, such that if all components in a set are assumed to be not working, there are no discrepancies between the observed behavior and the model-computed behavior. These sets are called diagnoses. The diagnoses can subsequently be used to determine further diagnostic actions, such as determining the best probing point.

2. Hierarchy in model-based diagnosis

Diagnostic algorithms based on best-first generation of diagnoses [8] have a

* This research is sponsored by SKBS, a Dutch foundation that stimulates cooperation of universities, industry and business on knowledge-based systems.

[1] We do not use fault models [18], or physical impossibility [3].

complexity of $O(n^i * f_m(n,i))$, where n is the number of components, i is the number of malfunctioning components, and $f_m(n,i)$ is an expression for the maximum amount of time it takes to determine whether a diagnosis of size i is consistent. $f_m(n,i)$ strongly depends on the behavior of the components in the model m. For digital circuits[2] $f_m(n,i) = n * 2^i$ while for circuits consisting of integer adders and multipliers $f_m(n,i) = \infty$ as the problem of determining consistency is undecidable for these models. A proof of this can be found in [16].

Hammink et al. [6] show that we can reduce this to $O(^d log(n) * d^i * f_m(d,i))$, if every component of a model in the hierarchy consists of exactly d lower level components, and we can determine a unique diagnosis at each level of the hierarchy[3].

The general algorithm for hierarchic model-based diagnosis in an n-level hierarchy is algorithm1.

ALGORITHM 1

(1) a:=n; Generate a set of diagnoses $D_{abstract}(a)$ for the most abstract model (level n).

(2) Transform $D_{abstract}(a)$ to a set possible diagnoses $D_{detailed}(a-1)$ for the detailed model.

(3) Check each possible diagnosis in $D_{detailed}(a-1)$ for consistency, giving a set of diagnoses $D_{checked}(a-1)$.

(4) Discriminate between the diagnoses in $D_{checked}(a-1)$ by making probes[4] until we are satisfied with the set. We call the result $D_{probed}(a-1)$.

(5) If a=2 we are finished, else $D_{abstract}(a-1) := D_{probed}(a-1), a := a-1$ and goto step 2.

We illustrate this algorithm with an example:

Example 1

Consider the hierarchic model depicted in fig. 1. At the highest level we have only one diagnosis, so $D_{abstract}(2) = \{[M]\}$. If we translate this to the more detailed level, we get the set of possible diagnoses $D_{detailed}(1) = \{[m_1], [m_2], [m_3], [m_1, m_2], [m_1, m_3], [m_2, m_3], [m_1, m_2, m_3]\}$.

If we now check these possible diagnoses for consistency, we see that the possible diagnosis $[m_3]$ is not a diagnosis as no matter what its output is, if m_1 and m_2 function correctly the output of m_2 must be at least 5. So $D_{checked}(1)$

[2] Consistency checking in digital circuits is NP-complete. The best *known* algorithms are exponential in the number of malfunctioning components.

[3] This complexity expression assumes that $f_m(d,i)$ is the same at every level of the hierarchy, which is not necessarily the case. For a discussion of this phenomenon, see [4]

[4] In the model-based diagnosis paradigm, discrimination is traditionally done by probing. No efficient algorithms exist at this moment for generating tests to discriminate between diagnoses.

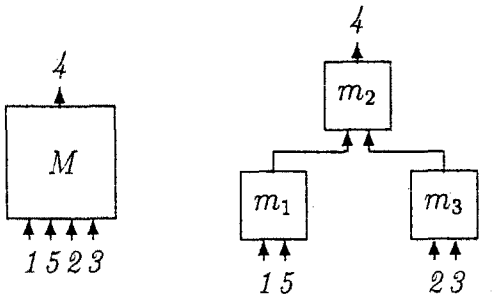

Fig. 1. A 4-input maximizer implemented by three input maximizers.

becomes $\{[m_1], [m_2], [m_1, m_2], [m_1, m_3], [m_2, m_3], [m_1, m_2, m_3]\}$. We can subsequently discriminate between these diagnoses after which the algorithm terminates. □

2.1. DESIRABLE PROPERTIES OF HIERARCHIC MODELS

Of course, some hierarchic models are better then others. We here give three desirable properties of hierarchic models.

Downward failure property: (Weld and Addanki [19]) If a hierarchy has this property, this means that diagnoses which are impossible at the abstract level are impossible at the detailed level as well. We can therefore use the results computed on the abstract level to reduce the search space on the detailed level. The gain of using hierarchy lies in this reduction. In other work on abstraction, similar notions to the downward failure property are used, for instance: TI-abstractions [5], upward solution property [10], the consistency condition C2 [15], C-reformulations [16].

Allow easy consistency checking: Checking the set of possible diagnoses for consistency (in step 3 of algorithm 1) can take much reasoning effort. It is therefore desirable to design hierarchies in such a way as to minimise this. In this paper we define a class of hierarchic models, called ID-hierarchies such that this step can be avoided.

Allow good discriminability: When we transform a high-level diagnosis (in step 2 of algorithm 1) the resulting set of low-level diagnoses can be very large. It is therefore desirable that we can discriminate between diagnoses (in step 4) as good as possible, retaining as few diagnoses as possible, preferably only one. In this paper we define a class of hierarchic models, called SD-hierarchies which ensures that we can find a single diagnosis at every level of the hierarchy at a reasonable cost.

2.2. CREATING HIERARCHIC MODELS

One of the main difficulties when using hierarchy in model-based diagnosis is creating the hierarchic models. In this paper, we create a hierarchic model from a

non-hierarchic model. For every model in the hierarchy, we have to specify two items:

(1) A structural model, showing how components are connected.

(2) A set of behavioral descriptions of the components.

As we will show later on in this paper, we only construct abstract structural models for our hierarchy. The behavioral functions of the components in abstract models are directly derived from the behavioral functions of the components in detailed models and are not abstracted.[5] We call these types of hierarchies *structural hierarchies.*

3. ID[6]-hierarchies.

In step 2 of algorithm 1, a set of diagnosis $D_{abstract}$ for an abstract model is transformed into a set of *possible* diagnoses $D_{detailed}$ for a detailed model. We subsequently (in step 3) check each of these possible diagnoses for consistency to see whether they are diagnoses for the detailed model, giving the set $D_{checked}$. This consistency checking can be a very time-consuming process [16].

In this section we discuss ID-hierarchies. ID-hierarchies have the property that if we translate a diagnosis from an abstract model to a set of possible diagnoses of a detailed model, every possible diagnosis in that set is also a diagnosis, so that $D_{checked} = D_{detailed}$. We illustrate this with an example:

Example 2

Consider again the hierarchic model given in fig. 1. As we have seen in example 1, $D_{checked} \neq D_{detailed}$ so this hierarchy is not an ID-hierarchy. If we now assume that M is a 4-input adder and m_1, m_2 and m_3 are two-input adders, the possible diagnosis $[m_3]$ is now a diagnosis. As all other possible diagnoses are diagnoses too, $D_{checked} = D_{detailed}$ so this model is an ID-hierarchy. □

Using ID-hierarchies has the advantage that step 3 of algorithm 1 can be avoided entirely as $D_{checked} = D_{detailed}$.

3.1. DEFINING ID-HIERARCHIES

In this section we define ID-hierarchies and prove that we can indeed avoid step 3 of algorithm 1. However, we must first define indistinguishability:

[5] Automatically abstracting behavioral functions is an extremely difficult problem [7, 12] and even though some work has been done in this direction [11] it is still mainly unsolved.

[6] ID stands for indistinguishability. This refers to the fact that fault effects of all detailed components in the same abstract component are indistinguishable from each other. This property is related to the concept of fault collapsing [14].

DEFINITION 1 (Indistinguishability)

A set of components C of a model is indistinguishable iff for every possible combination of values of the inputs and outputs of the model, every minimal conflict contains either:

(1) all components in C, or

(2) no components from C

We call a set maximally indistinguishable if no strict superset of that set is indistinguishable. □

Finding *all* possible minimal conflicts is an undecidable problem, and even in simple cases computationally very expensive [13]. In [1] a method is described to find all possible conflicts which can be found by constraint propagation [9], the so-called GDE-conflicts. From here on we will use only indistinguishability w.r.t. GDE-conflicts rather than all conflicts.

DEFINITION 2 (Two-level ID-hierarchy)

Let $M_{detailed}$, the detailed model, and $M_{abstract}$, the abstract model, be two models. $M_{detailed}$ and $M_{abstract}$ form an ID-hierarchy iff $M_{abstract}$ can be constructed from $M_{detailed}$ such that every component in $M_{abstract}$ is constructed by combining a maximally indistinguishable set of $M_{detailed}$. □

We can use these definitions to prove the following theorem which shows the utility of ID-hierarchies:

THEOREM 1

Let M_A and M_D be the abstract model and the detailed model of an ID-hierarchy. For every possible combination of values of the inputs and outputs holds that for every diagnosis M_A, if we transform it to a set of possible diagnoses for M_D, that every possible diagnosis is a diagnosis for M_D.

Proof

Consider the minimal conflicts of M_A. As the components of M_A are maximally indistinguishable sets, if we replace every component in each conflict by its corresponding components of M_D, the resulting set consists of precisely all minimal conflicts of M_D. As both models have the same minimal conflicts, they must have the same minimal diagnoses as well. If both models have the same minimal diagnoses, they must have the same diagnoses as well. □

3.2. GENERATING TWO-LEVEL ID-HIERARCHIES

In this section, we present an algorithm which generates an abstract model from a detailed model such that the resulting two-level hierarchy is an ID-hierarchy. The simplest algorithm for determining maximally indistinguishable sets (and therefore an ID-hierarchy) is to first determine all possible conflicts and then construct the maximally indistinguishable sets from these possible conflicts. Unfortunately the number of possible conflicts is $O(2^n)$ where n is the number of components in the model, and therefore any algorithm to find them all has a time complexity of at least $O(2^n)$. The algorithm we present (algorithm 2) finds the indistinguishable sets in $O(n)$ time. The proof that this algorithm actually generates the indistinguishable sets (and sometimes maximally indistinguishable sets) is quite involved and can be found in [17].

The algorithm uses as input a (detailed) model and information about the invertibility[7] of the behavior of the components of that model, and generates as output a second (abstract) model. This algorithm is only useful for generating ID-hierarchies from combinational models as it makes a sharp distinction between components before and after a given component in the input-output ordering.

Let $C_{detailed}$ be the set of components of the detailed model. The algorithm then generates the set $C_{abstract}$ of indistinguishable subsets of $C_{detailed}$. Each element of $C_{abstract}$ corresponds to an abstract component.

For each component $c \in C_{detailed}$ we define the following two sets:

Pre(c), which denotes direct predecessors of c (components whose output is directly connected to the inputs of c)

Post(c), which denotes all postdecessors of c (components whose inputs are influenced by the output of c, either directly or through other components).

ALGORITHM 2

(1) $C_{abstract} := \{\{c\} \mid c \in C_{detailed} \wedge$ one of the outputs of c is a system output$\}$.

(2) Take a component c from a set $S \in C_{abstract}$ such that there exists a component $v \in Pre(c)$ which is not yet part of any set in $C_{abstract}$.

(3) If $Post(c) \cup \{c\} = Post(v)$ and c is invertible then $S := S \cup \{v\}$ else $C_{abstract} := C_{abstract} \cup \{v\}$.

(4) If not all components have been clustered goto 2.

We illustrate this algorithm with the following example:

Example 3

In this example we will illustrate algorithm 2 by showing how the abstract

[7] We call a component invertible if, for every input of that component, we can infer the value of that input, given the values of all other inputs and outputs of that component.

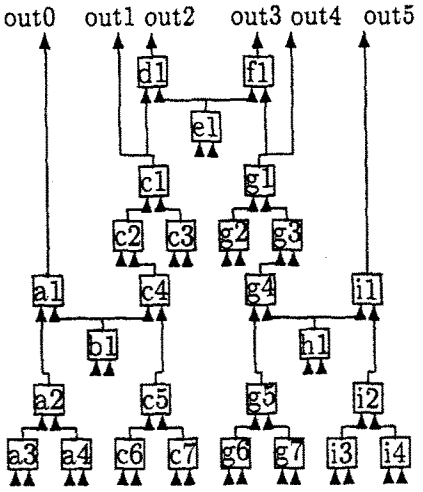

Fig. 2. Detailed model.

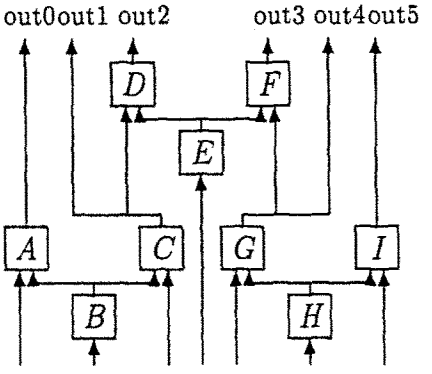

Fig. 3. Abstract model.

model[8] of fig. 3 can be constructed from the detailed model in fig. 2. In the detailed model all components are assumed to be invertible.

Step 1: In this step we collect all components whose output is a system output in $C_{abstract}$, so $C_{abstract}$ becomes $\{\{a1\},\{c1\},\{d1\},\{f1\},\{g1\},\{i1\}\}$.

Step 2: In this step we have to take a component from a set in $C_{abstract}$ such that it has at least one predecessor not in $C_{abstract}$. We choose here $d1$ with predecessor $e1$.

Step 3: As $\text{Post}(d1) \cup \{d1\} = \{d1\}$ and $\text{Post}(e1) = \{d1, f1\}$, component $e1$ becomes part of a new abstract component so $C_{abstract} := \{\{a1\},\{c1\},\{d1\},\{e1\},\{f1\},\{g1\},\{i1\}\}$.

Step 4: Goto Step2, as not all components are clustered.

If, in step 2, we had chosen $c1$ with predecessor $c2$, then we would have found in Step 3 that as $c1$ is invertible and $\text{Post}(c1) \cup \{c1\} = \text{Post}(c2)$, $c1$ and $c2$ become part of the same abstract component so $C_{abstract} := \{\{a1\},\{c1, c2\},\{d1\},\{f1\},\{g1\},\{i1\}\}$.

The algorithm iterates until $C_{abstract} = \{\{a_1, \ldots, a_4\}, \ldots, \{i_1, \ldots, i_4\}\}$ has been obtained. □

It is not hard to see that the algorithm runs in $O(n)$ time with n the number of components in the detailed model. The main drawback lies in its heavy reliance on invertibility of the components; non-invertible components tend to increase the number of abstract components, as we show in the following example:

[8] For readability, we have deleted several inputs from the abstract model.

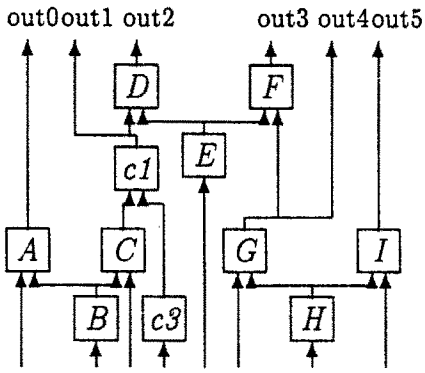

Fig. 4. c1 is non-invertible.

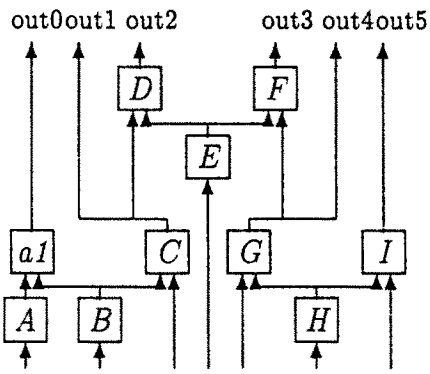

Fig. 5. a1 is non-invertible.

Example 4

In fig. 4 we show the model which results from applying algorithm 2 to the model in fig. 2 if component c1 is non-invertible (and all other components are invertible). In this model c1 and c3 are "split off" from *C*, so *C* now consists of c2 and c4-c7.

If component a1 from the model in fig. 2 is non-invertible, the model in fig. 5 results. Here component a1 is "split off" from *A*, so *A* now consists of a2-a4. □

If we have a large number of non-invertible components, we can get an abstract model which is almost the same as the detailed model. In this situation, hierarchic diagnosis using ID-hierarchies can have worse performance than diagnosis using only the detailed model.

3.3. GENERATING MULTI-LEVEL ID-HIERARCHIES

Algorithm 2 takes a detailed model as input and generates an abstract level as output, forming a two level ID-hierarchy. We can extend this method towards a multi-level ID-hierarchy using the concept of classes of probing costs [6].

3.3.1. Classes of probing costs

A physical system contains probing points where measurements can be made to discriminate between possible diagnoses. Often the cost of probing these points is not equal for all probing points; some are physically hard to reach, some require expensive probing equipment, some carry signals which are difficult to interpret etc. We suppose that an estimate of the cost of measurement is available and use this estimate to collect probing points with more or less equal costs of measurement in classes of probing costs (CPCs). The size and number of these classes depends on

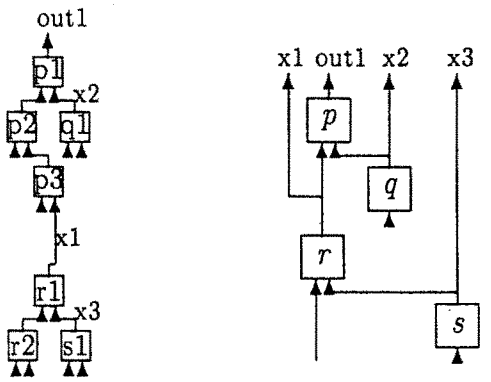

Fig. 6. An intermediate level in an ID-hierarchy.

the application being diagnosed. The class with the lowest cost are always the outputs.

3.3.2. Generating multi-level ID-hierarchies

Using CPCs we can build a multi-level ID-hierarchy. Suppose we have n CPCs, labeled $CPC_1 \ldots CPC_n$ with CPC_n being the "cheapest" class. We already have the detailed level (level 1) of the hierarchy and we can use algorithm 2 to create level n (the most abstract level). Note that in step 1 of algorithm 2 we use the outputs of the system CPC_n to determine the starting clusters.

To create level $n-1$ of the hierarchy we use algorithm 2 again, but this time we take the set $CPC_n \cup CPC_{n-1}$ to be the outputs of the system. Similarly, to create level $n-2$, we use $CPC_n \cup CPC_{n-1} \cup CPC_{n-2}$ etcetera. For level 1 we would use all connections, this means that in step 1 every component becomes a starting cluster so the algorithm terminates immediately. We illustrate this with an example:

Example 5

In the leftmost model of fig. 6, we divide the probing points in three *CPC*s. $CPC_3 = \{\text{out1}\}$, $CPC_2 = \{x1, x2, x3\}$, and CPC_1 contains all other connections. As we are constructing level 2 of the hierarchy, we assume $CPC_3 \cup CPC_2$ to be the system outputs, and after executing algorithm 2, the rightmost model of fig. 6 results. □

3.4. HOW GOOD ARE ID-HIERARCHIES

Which of the desirable properties mentioned earlier do ID-hierarchies have?

Downward failure property: To create ID-hierarchies from a detailed model, we use only two operations, grouping components and removing connections. Mozetic [15]

proves that if we only allow these operations, resulting hierarchic models obey the consistency condition, which is stronger then the downward failure property.

Allow easy consistency checking: No consistency checking is needed. ID-hierarchies are constructed in such a way that every possible diagnosis is consistent and therefore a diagnosis.

Allow good discriminability: It is not always possible to determine a unique diagnosis at every level of the hierarchy at a reasonable cost. The probing points possibly needed to determine a unique diagnosis can have arbitrary cost as the following example shows:

Example 6

Suppose that, in the model of fig. 2, we have probed the output of component i2 and determined it to be erroneous. This means that one of the components $i2$, $i3$ or $i4$ must be malfunctioning. If the outputs of $i3$ and $i4$ are both very difficult to measure, we cannot determine the unique diagnosis cheaply. □

As we shall see in the next section, a possible solution to this problem is the use of SD-hierarchies.

4. SD[9]-hierarchies

In step 4 of algorithm 1, we discriminate between a set of diagnoses $D_{checked}$ and, as seen in example 6, the cost of determining an unique, minimal diagnosis can be arbitrarily high. Yet the complexity of hierarchical model-based diagnosis is $O({}^{d}log(n) * (dp)^{i} * f_m(n,i))$ if p is the number of diagnoses found on each level of the hierarchy. In this expression, n can be very large, but d and i are usually very small, so we want p to be as small as possible (preferably 1).

In this section we will discuss SD-hierarchies. SD-hierarchies can be informally defined as being structured in such a way that we can always find a single, minimal diagnosis using only probing points cheaper than some predetermined cost.

4.1. GENERATING TWO-LEVEL SD-HIERARCHIES

To determine the single, minimal diagnosis we sometimes have to probe very expensive probing points. SD-hierarchies are constructed in such a way that this can not occur. This is done by constructing models which have only probing points which can be measured relatively cheap, all other connections are removed from the model.

[9] SD stands for single diagnosis.

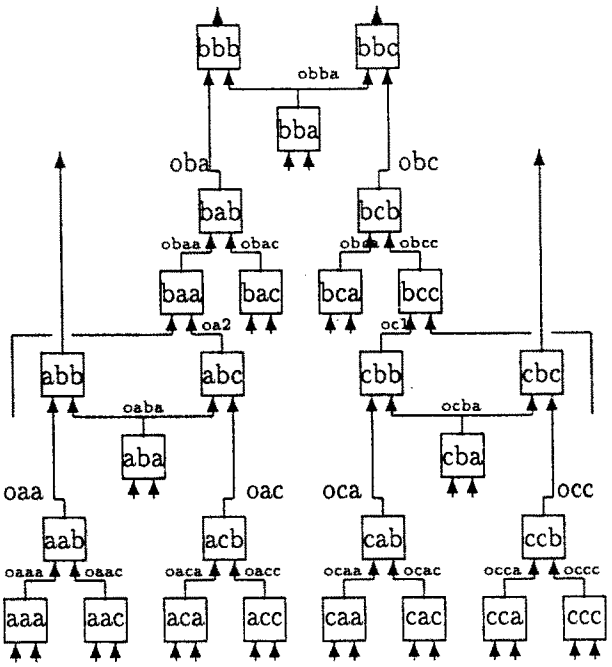

Fig. 7. Most detailed model of a system.

ALGORITHM 3

(1) If two components are only connected by 'expensive' connections, collapse them to one component.

(2) Remove all 'expensive' connections.

This algorithm runs in linear time. As the only connections remaining in the abstract model are all 'cheap' connections, we can determine a single minimal diagnosis of the abstract model by probing only 'cheap' connections. The deeper we descend in the hierarchy, the more expensive probing becomes.

4.2. GENERATING MULTI-LEVEL SD-HIERARCHIES

We can generate a multi-level SD-hierarchy using SD-abstractions and classes of probing costs in a method similar to that described in section 3.3. The most abstract model would have all but the cheapest class of connections removed, the next most abstract model would have all but the cheapest two classes removed etcetera. Again, we will illustrate this with an example:

Example 7

In fig. 7 a detailed level model is depicted. This model has four CPCs:

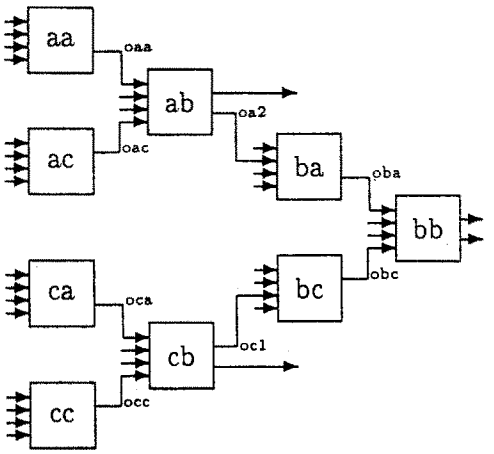

Fig. 8. VeryExpensive removed.

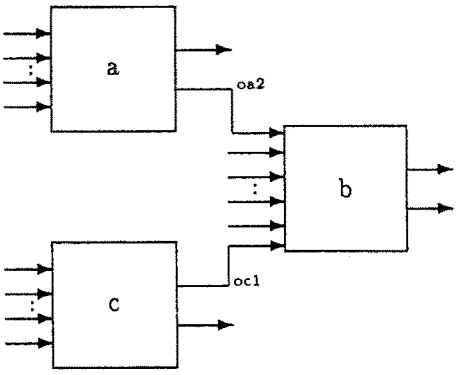

Fig. 9. Expensive removed.

Very expensive: the connections labeled with four letters, e.g. oaca.
Expensive: the set $\{oaa, oac, oca, occ, oba, obc\}$.
Cheap: the set $\{oa2, ocl\}$.
Very Cheap: the system inputs and outputs.

If we remove the VeryExpensive connections from the model in fig. 7 the model in fig. 8 results. If we subsequently remove the Expensive connections, the model in fig. 9 results. We can also remove the Cheap connections from the model in fig. 9 in which case a model results which consists of one component with 28-inputs and 4 outputs. □

4.3. HOW GOOD ARE SD-HIERARCHIES

Which of the desirable properties mentioned earlier do SD-hierarchies have?

Downward failure property: As with ID-hierarchies we only group components and remove connections, so SD-hierarchies have the downward failure property.

Allow easy consistency checking: SD-hierarchies are **not** necessarily ID-hierarchies. Therefore all possible diagnoses must be checked for consistency.

Allow good discriminability: In SD-hierarchies we can always determine the unique diagnosis at a given level, using only probing points from the corresponding and cheaper CPCs, so at relatively high levels we use only relatively cheap probing points. The more expensive probing points are only used at the more detailed levels.

5. Conclusions

We have given a general algorithm for hierarchical diagnosis. This algorithm

has two potentially very costly operations: checking diagnoses for consistency and discriminating between diagnoses. We have defined two classes of hierarchic models designed to deal with these problems:

- ID-hierarchies which make consistency checking redundant.
- SD-hierarchies which ensure that only relatively cheap probing points are used to determine a single, minimal diagnosis.

We have given algorithms to create these hierarchies. Both these algorithms run in linear time.

Unfortunately, SD-hierarchies are not necessarily ID-hierarchies (or vice versa), so we can not combine their beneficial effects. For a given application we therefore have to decide whether an SD-hierarchic model or an ID-hierarchic model is better. In some cases, we can make this choice directly, e.g.:

- The application contains many non-invertible components. In this case, using ID-hierarchies is not a good idea.
- All probing points in the application cost more or less the same. In this case SD-hierarchies are less useful.

In other cases, experiments will have to decide.

References

[1] R.R. Bakker, in: *Knowledge-Based Diagnosis of Technical Systems, a Three Year Progress Report*, University of Twente, Enschede (1991).

[2] R. Davis, Diagnostic reasoning based on structure and behavior, Artificial Intelligence 24 (1984) 347–410.

[3] G. Friedrich, G. Gottlob and W. Nejdl, Physical impossibility instead of fault models, in: *Proc. 8th Nat. Conf. on Artificial Intelligence*, Boston, MA, 1990 (AAAI Press/MIT Press, Menlo Park, CA, 1990) pp. 331–336.

[4] M.R. Genesereth, The use of design descriptions in automatic diagnosis, Artificial Intelligence 24 (1984) 411–436.

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

[6] E. Hammink, D-J. Out and R.R. Bakker, Controlled reasoning and hierarchy in model-based diagnosis, in: *Proc. 10th Conf. on Second Generation Expert Systems*, Avignon, 1991, pp. 217–230.

[7] W. Hamscher, Model-based troubleshooting of digital systems, MIT Technical Report AI-TR 1074 (MIT Press, Cambridge, MA, 1988).

[8] J. de Kleer, Focusing on probable diagnoses, in: *Proc. 8th Nat. Conf. on Artificial Intelligence (AAAI-91)* (AAAI Press/MIT Press, Menlo Park, 1991) pp. 842–848.

[9] J. de Kleer and B.C. Williams, Diagnosing multiple faults, Artificial Intelligence 32 (1987) 97–130.

[10] C.A. Knoblock, J.D. Tenenberg and Q. Yang, A spectrum of abstraction hierarchies for planning, in: *Working Notes of the AGAA-90 Workshop*, Boston, 1990, pp. 24–35.

[11] A.N. Kumar and S.J. Upadhyaya, Automating representation, in: *Working Papers of the Second AAAI Workshop on Model Based Reasoning*, Boston, 1990, pp. 124–130.

[12] R. Lbath, N. Giambiasi and C. Delorme, FRaHM : Framework for reasoning about hierarchical/ multi-vues models, in: *System Fault Diagnostics, Reliability and Related Knowledge-Based Approaches*, Vol. 2, eds. S. Tzafestas, M. Singh and G. Schmidt (Reidel, 1987) pp. 29–41.
[13] A. Makarovic, Parsimony in model-based reasoning, University of Twente, Ph.D. Thesis, Enschede (1991).
[14] E.J. McCluskey and F.W. Clegg, Fault equivalence in combinational logic networks, IEEE Trans. Comp. C-20 (1971) 1286–1293.
[15] I. Mozetič, Hierarchical model-based diagnosis, Int. J. Man-Machine Stud. 35 (1991) 329–362.
[16] D-J. Out and R.R. Bakker, An analysis of model-based diagnosis using problem reformulation, University of Twente, Memoranda Informatica 92-03 (1992).
[17] D-J. Out, Strategies for efficient model-based troubleshooting, University of Twente, Ph.D. Thesis, Enschede (1993).
[18] P. Struss and O. Dressler, Physical negation, integrating fault models into the general diagnostic engine, in: *Proc. 11th Int. Joint Conf. on Artificial Intelligence*, Detroit, MI, 1989, ed. N.S. Sridharan (Morgan Kaufmann, San Mateo, CA, 1989) pp. 1318–1323.
[19] D.S. Weld and S. Addanki, Task-driven model abstraction, University of Washington (1990).