The Lydia Approach to Combinational Model-Based Diagnosis

Alexander Feldman * **Gregory Provan** ** **Arjan van Gemund** *

* *Delft University of Technology, Delft, The Netherlands (email: {a.b.feldman,a.j.c.vangemund}@tudelft.nl)*
** *University College Cork, Cork, Ireland (email: g.provan@cs.ucc.ie)*

Abstract: Lydia is a framework for Model-Based Diagnosis (MBD). We have used the Safari diagnostic reasoner (part of Lydia) to solve the synthetic track problems in the DXC'09 diagnostic challenge. In this paper we describe our approach.

Keywords: model-based diagnosis; diagnostic competition; DXC; DXC'09; Lydia; Safari.

1. INTRODUCTION

We consider the first diagnostic competition (DXC'09) (Kurtoglu et al., 2009) an important milestone in applying techniques for automated diagnosis in practice. For example, the synthetic track allows for "stress-testing" of diagnostic algorithms using "difficult" and boundary cases (Feldman et al., 2008a), while the industrial track forces algorithm designers to have a better view on the technicalities of applying automated diagnosis in practice. Although many of those technical aspects are deemed "easy" to solve, the overall task of diagnosis is challenging and DXC'09 is a right step towards better understanding diagnosis (and related fields like testing and repair) in general.

Lydia (Feldman et al., 2006) is a declarative modeling language specifically developed for Model-Based Diagnosis (MBD). The language core is propositional logic, enhanced with a number of syntactic extensions for ease of modeling. The accompanying toolset currently comprises a number of diagnostic engines and a simulator tool.

Safari (Feldman et al., 2008b) (StochAstic Fault diagnosis AlgoRIthm) computes minimal diagnoses while sacrificing guarantees of optimality, but for diagnostic systems in which faults are described in terms of an arbitrary deviation from nominal behavior, Safari can compute diagnoses several orders of magnitude faster than competing algorithms, which is empirically confirmed by DXC'09.

We had planned to participate with our diagnostic framework Lydia in both the synthetic and industrial tracks of DXC'09. Unfortunately, the industrial track required a significant amount of modeling effort (our framework is Model-Based (de Kleer and Williams, 1987)), and due to lack of time (and resources) we have submitted Lydia in the synthetic track only, where the modeling effort could be easily automated. We plan to finish the modeling work and to evaluate Lydia in the industrial track internally (and to participate in follow-up industrial track competitions).

In this paper we describe the approach we take to solving the synthetic track in the DXC'09 competition. We have used the Safari algorithm, which we explain in this paper. We show the results in terms of the diagnostic metrics that have been chosen by the DXC'09 organizers.

Note that Sec. 2 and Sec. 3 are also in (Feldman et al., 2008a), except the part describing the difference of Safari from "standard" MBD algorithms. We have included these sections to make our paper self-contained.

2. TECHNICAL BACKGROUND

This section formalizes some standard MBD notions, and explains, on an intuitive level, the Safari inference algorithm. It is important to note that Safari performs inference in a manner that is different to "standard" MBD algorithms, such as GDE (de Kleer and Williams, 1987) or the ATMS (De Kleer, 1986), in that it interleaves stochastic search with SAT consistency-checking of potential diagnoses, as we will explain.

2.1 Diagnosis and Minimal Diagnosis

We adopt the traditional diagnostic definitions (de Kleer and Williams, 1987), except that we use propositional logic terms (conjunctions of literals) instead of sets of failing components.

Central to MBD, a *model* of an artifact is represented as a propositional **Wff** over some set of variables. Discerning two subsets of these variables as *assumable* and *observable* [1] variables gives us a diagnostic system.

Definition 1. (Diagnostic System). A diagnostic system DS is defined as the triple DS $= \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, where SD is a propositional theory over a set of variables V, COMPS $\subseteq V$, OBS $\subseteq V$, COMPS is the set of assumables, and OBS is the set of observables.

[1] In the MBD literature the assumable variables are also referred to as "component", "failure-mode", or "health" variables. Observable variables are also called "measurable", or "control" variables.

Throughout this paper we assume that $\mathrm{OBS} \cap \mathrm{COMPS} = \emptyset$ and $\mathrm{SD} \not\models \perp$. Although SAFARI delivers good results for a larger class of diagnostic models, this paper focuses on the well-known weak-fault models[2].

Definition 2. (Weak-Fault Model). A diagnostic system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$ belongs to the class **WFM** iff SD is in the form $(h_1 \Rightarrow F_1) \wedge \ldots \wedge (h_n \Rightarrow F_n)$ such that $1 \leq i, j \leq n$, $\{h_i\} \subseteq \mathrm{COMPS}$, $F_j \in \mathbf{Wff}$, and none of h_i appears in F_j.

Note the conventional selection of the sign of the "health" variables $h_1, h_2, \ldots, h_n$. Other authors use "ab" for abnormal or "ok" for healthy. Weak-fault models are sometimes referred to as models with *ignorance of abnormal behavior* (de Kleer et al., 1992), or *implicit fault systems*. The traditional query in MBD computes terms of assumable variables which are explanations for the system description and an observation.

Definition 3. (Health Assignment). Given a diagnostic system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, an assignment HA to all variables in COMPS is defined as a health assignment.

A health assignment HA is a conjunction of propositional literals. In some cases it is convenient to use the set of negative or positive literals in HA. These two sets are denoted as $Lit^-(\mathrm{HA})$ and $Lit^+(\mathrm{HA})$, respectively.

What follows is a formal definition of consistency-based diagnosis.

Definition 4. (Diagnosis). Given a diagnostic system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, an observation α over some variables in OBS, and a health assignment ω, ω is a diagnosis iff $\mathrm{SD} \wedge \alpha \wedge \omega \not\models \perp$.

In the MBD literature, a range of types of "preferred" diagnosis has been proposed. This turns the MBD problem into an optimization problem. In the following definition we consider the common subset-ordering.

Definition 5. (Minimal Diagnosis). A diagnosis ω is minimal if no diagnosis ω' exists such that $Lit^-(\omega') \subset Lit^-(\omega)$.

The set of all minimal diagnoses characterizes all diagnoses given a weak-fault model, but that does not hold in general (de Kleer et al., 1992). With no restrictions on the model, faulty components may "exonerate" each other, resulting in a health assignment containing a proper superset of the negative literals of another diagnosis not to be a diagnosis.

Diagnosis cardinality gives us another partial ordering: a diagnosis is defined as *minimal cardinality* iff it minimizes the number of negative literals.

Definition 6. (Cardinality of a Diagnosis). The cardinality of a diagnosis, denoted as $|\omega|$, is defined as the number of negative literals in ω.

A minimal cardinality diagnosis is a minimal diagnosis, but the opposite does not hold. There are minimal diagnoses which are not minimal cardinality diagnoses.

[2] DXC'09 provides the topology and nominal behavior of the ISCAS85 circuits only. Participants may assume "stuck-at" behavior but we have used weak-fault models only.

2.2 Diagnostic Inference: "Traditional" versus SAFARI *Methodologies*

This section provides an intuitive comparison of SAFARI with "standard" MBD algorithms, in order to clarify our novel algorithm, which will be explained in more detail in the following section. SAFARI performs inference in a manner that is different than "standard" MBD algorithms. In the following, we first summarize the standard MBD approach, and contrast it with that of SAFARI.

Given an observation OBS denoting an abnormal condition (or symptom), a standard MBD algorithm A performs diagnosis in a two-step process. In the first step, A computes the conflicts for OBS, i.e., a conflict is a set of components which cannot all be operating properly given a symptom. We denote a conflict γ as an assignment to a subset of health variables.

Definition 7. (Conflict). Given a diagnostic system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, and an observation α over some variables in OBS, γ is a conflict iff $\mathrm{SD} \wedge \alpha \wedge \gamma \models \perp$.

Most MBD approaches will compute minimal conflicts, i.e., conflicts which are minimal with respect to some preference criterion ϕ. Both subset- and cardinality-minimal conflicts have been studied in the literature. Conflicts are typically computed using constraint-propagation methods (de Kleer and Williams, 1987).

In the second step, a standard MBD algorithm A computes, from a set of conflicts γ, a diagnosis (or preferred diagnosis, using some preference function ϕ). A diagnosis is a health assignment to each system component, as defined before. Diagnoses (or minimal diagnoses with respect to ϕ) are computed using some logic-based consistency-checking mechanism, such as a SAT solver (McAllester, 1990) or through computing a minimal hitting set of the minimal conflicts (de Kleer and Williams, 1987). The minimal conflicts can be computed by, for example, the ATMS (De Kleer, 1986).

In contrast, SAFARI performs diagnostic inference as follows. SAFARI avoids the conflict-analysis phase by (a) guessing an initial diagnosis given an observation α, and then (b) trying to reduce the cardinality of this initial diagnosis ω through flipping the values of some health variables in ω from faulty to healthy, performing a consistency-check after each flip. SAFARI uses an incomplete SAT-solver, BCP, for each consistency-check (McAllester, 1990). This randomized search, in conjunction with the computationally efficient consistency-checking, is what distinguishes SAFARI from traditional algorithms, and also what gives it its computational advantages over traditional algorithms.

3. STOCHASTIC MBD ALGORITHM

This section presents a more formal specification of SAFARI, an algorithm for computing multiple-fault diagnoses using stochastic search. As described earlier, SAFARI uses a two-step diagnostic process. Step 1 involves randomly choosing candidates. Step 2 attempts to minimize the fault cardinality of these candidates.

Algorithm 1 shows the top-level pseudocode for SAFARI. Step 1 takes place in line 4 of Algorithm 1; the remainder of the pseudocode of Algorithm 1 performs Step 2.

Algorithm 1 uses a number of utility functions, which we briefly review; see (Feldman et al., 2008a) for details. The ImproveDiagnosis subroutine takes a term as an argument and changes the sign of a random negative literal. If there are no negative literals, the function returns the original argument. The implementation of RandomDiagnosis uses a modified DPLL solver (Davis et al., 1962) returning a random SAT solution of SD $\wedge\, \alpha$.

Similar to deterministic methods for MBD, Safari uses a SAT-based procedure for checking the consistency of SD $\wedge\, \alpha \wedge \omega$. Because SD $\wedge\, \alpha$ does not change during the search, the incremental nature of the Logic-Based Truth Maintenance System (LTMS) assumption checking (McAllester, 1990) greatly improves the search efficiency. The implementation of Safari combines a BCP-based LTMS to check for inconsistencies. If a candidate is consistent, a subsequent DPLL-based check is invoked for completeness.

The randomized search process performed by Safari has two parameters, M and N. There are N independent searches that start from randomly generated starting points. The algorithm tries to improve the cardinality of the initial diagnoses (while preserving their consistency) by randomly "flipping" fault literals. The change of a sign of literal is done in one direction only: from faulty to healthy.

Algorithm 1 Safari: A greedy stochastic hill climbing algorithm for approximating the set of minimal diagnoses.

```
 1: function HillClimb(DS, $\alpha$, $M$, $N$) returns a trie
        inputs: DS = $\langle$SD, COMPS, OBS$\rangle$, diag. system
                $\alpha$, term, observation
                $M$, integer, climb restart limit
                $N$, integer, number of tries
 2:     $n \leftarrow 0$
 3:     while $n < N$ do
 4:         $\omega \leftarrow$ RandomDiagnosis(SD, $\alpha$)
 5:         $m \leftarrow 0$
 6:         while $m < M$ do
 7:             $\omega' \leftarrow$ ImproveDiagnosis($\omega$, $\rho$)
 8:             if SD $\wedge\, \alpha \wedge \omega' \not\models \perp$ then
 9:                 $\omega \leftarrow \omega'$
10:                 $m \leftarrow 0$
11:             else
12:                 $m \leftarrow m + 1$
13:             end if
14:         end while
15:         unless IsSubsumed($R$, $\omega$) then
16:             AddToTrie($R$, $\omega$)
17:             RemoveSubsumed($R$, $\omega$)
18:         end unless
19:         $n \leftarrow n + 1$
20:     end while
21:     return $R$
22: end function
```

Each attempt to find a minimal diagnosis terminates after M unsuccessful attempts to "improve" the current diagnosis stored in ω. Thus, increasing M will lead to a better exploitation of the search space and, possibly, to diagnoses of lower cardinality, while decreasing it will improve the overall speed of the algorithm.

It is possible that two diagnostic searches may result in the same minimal diagnosis. To prevent this, we store the generated diagnoses in a trie R (Forbus and de Kleer, 1993), from which it is straightforward to extract the resulting diagnoses by recursively visiting its nodes. A diagnosis ω is added to the trie R by the function AddToTrie, iff no subsuming diagnosis is contained in R (the IsSubsumed subroutine checks on that condition). After adding a diagnosis ω to the resulting trie R, all diagnoses contained in R and subsumed by ω are removed by a call to RemoveSubsumed.

3.1 A Simple Example

We will use the Boolean circuit shown in Fig. 1 as a running example for illustrating Safari. The subtractor, shown there, consists of seven components: an inverter, two or-gates, two xor-gates, and two and-gates. The expression $h \Rightarrow (o \Leftrightarrow \neg i)$ models the normative (healthy) behavior of an inverter, where the variables i, o, and h represent input, output and health respectively. Similarly, an and-gate is modeled as $h \Rightarrow (o \Leftrightarrow i_1 \wedge i_2)$ and an or-gate by $h \Rightarrow (o \Leftrightarrow i_1 \vee i_2)$. Finally, an xor-gate is specified as $h \Rightarrow [o \Leftrightarrow \neg\,(i_1 \Leftrightarrow i_2)]$.

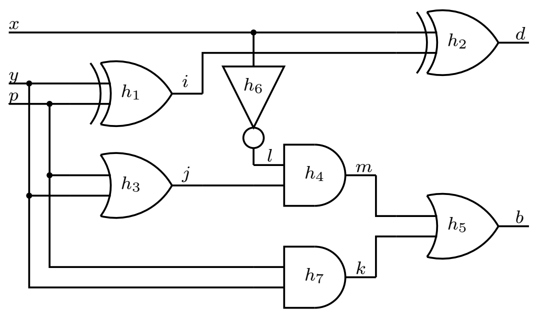

Fig. 1. A subtractor circuit

The above propositional formulae are copied for each gate in Fig. 1 and their variables renamed in such a way as to properly connect the circuit and disambiguate the assumables, thus obtaining a propositional formula for the Boolean subtractor, given by:

$$\text{SD} = \begin{cases} h_1 \Rightarrow [i \Leftrightarrow \neg\,(y \Leftrightarrow p)] \\ h_2 \Rightarrow [d \Leftrightarrow \neg\,(x \Leftrightarrow i)] \\ h_3 \Rightarrow (j \Leftrightarrow y \vee p) \\ h_4 \Rightarrow (m \Leftrightarrow l \wedge j) \\ h_5 \Rightarrow (b \Leftrightarrow m \vee k) \\ h_6 \Rightarrow (x \Leftrightarrow \neg l) \\ h_7 \Rightarrow (k \Leftrightarrow y \wedge p) \end{cases} \qquad (1)$$

The set of assumables is COMPS $= \{h_1, h_2, \ldots, h_7\}$ and the set of observable variables is OBS $= \{x, y, p, d, b\}$.

Diagnostic inference with Safari proceeds as follows. In step 1, the stochastic diagnostic search for the subtractor example will start from a random quintuple candidate [3]. In this particular version of our algorithm, once a component is marked as healthy, it cannot be changed back to faulty. To compensate for that, we perform multiple restarts from a random candidate. In our subtractor example and for

[3] (Feldman et al., 2008a) describe a method for determining the initial candidates.

$\alpha_3 = x \wedge y \wedge p \wedge \neg d \wedge \neg b$, if $h_1 \wedge h_2$ is in an initial "guessed" candidate, it will prove inconsistent with SD $\wedge$ α_3 and another quintuple fault candidate will be guessed.

Assume that this second candidate is $\omega_6 = \neg h_1 \wedge \neg h_2 \wedge h_3 \wedge \neg h_4 \wedge \neg h_5 \wedge h_6 \wedge \neg h_7$. Clearly, $\text{SD} \wedge \alpha_3 \wedge \omega_6 \not\models \perp$. The search algorithm may next try to improve the diagnosis by "flipping" the sign of h_7. The candidate $\omega_7 = \neg h_1 \wedge \neg h_2 \wedge h_3 \wedge \neg h_4 \wedge \neg h_5 \wedge h_6 \wedge h_7$ is a valid quadruple fault diagnosis and it can be improved twice more by "flipping" h_2 and h_5. This gives us the final double-fault $\omega_8 = \neg h_1 \wedge h_2 \wedge h_3 \wedge \neg h_4 \wedge h_5 \wedge h_6 \wedge h_7$. The actual algorithm is somewhat more involved as during the variable flipping it is normal to find inconsistencies. Instead of restarting, it will simply discard these inconsistent candidates until some termination criterion is satisfied.

Intuitively, from our example, due to the large number of double fault diagnoses explaining the same observation, it is not difficult to randomly guess sequences of variables which need to be false in order to explain the observation.

4. EXPERIMENTAL RESULTS

The DXC'09 synthetic track consists of the benchmark models of ISCAS85 circuits ((Brglez and Fujiwara, 1985)). These circuits are combinational, i.e., they contain no flip-flops or other memory elements. Note that the high-level structure of the ISCAS85 circuits, which can be beneficial to MBD analysis, has been flattened out. A reverse engineering effort had resulted in high-level Verilog models ((Hansen et al., 1999)). Table 1 summarizes the circuits used in the synthetic DXC'09 track.

The size of the circuits in Table 1 can be reduced by using cones (Siddiqi and Huang, 2007) for computing single-component ambiguity groups (Kurtoglu et al., 2009). The number of components in these smaller circuits is shown in the rightmost column of Table 1. To eliminate the need of expanding diagnoses containing faulty components inside cones, we have used the original (non-reduced) circuits. In future competitions, we are planning to use reduced models.

We have configured Safari with $M = |\text{COMPS}|$ and $N = 10$. The value of N we have computed empirically. Increasing N to 20 or decreasing it showed no significant change in M_{da}. We have also switched DPLL checking off. As a result of the BCP incompleteness, Safari produces inconsistent candidates, but this is rare (we have estimated less than 20% such candidates) and the overall effect on the metrics is positive.

We next define the metrics used in the synthetic track of DXC'09. For more information on the metrics as well as a broader discussion of their applicability cf. (Kurtoglu et al., 2009). The two computational metrics M_{cpu} and M_{mem} are straightforward: M_{cpu} is the total amount of busy CPU time spent by Safari and M_{mem} is the peak amount of allocated memory.

The next metric M_{ia} (classification errors) is defined as follows:

$$M_{\text{ia}} = \sum_{c \in \text{COMPS}} \frac{\sum_{\omega \in \Omega} m_{\text{err}}(c, \omega, \omega^*)}{|\Omega| \cdot |\text{COMPS}|} \tag{2}$$

	original					reduced
sys	|IN|	|OUT|	|COMPS|	V	C	|COMPS|
74182	9	5	19	47	75	6
74L85	11	3	33	77	118	15
74283	9	5	36	81	122	14
74181	14	8	65	144	228	15
c432	36	7	160	356	1 028	59
c499	41	32	202	445	1 428	58
c880	60	26	383	826	2 224	77
c1355	41	32	546	1 133	3 220	58
c1908	33	25	880	1 793	4 756	160
c2670	233	140	1 193	2 695	6 538	167
c3540	50	22	1 669	3 388	9 216	353
c5315	178	123	2 307	4 792	13 386	385
c6288	32	32	2 416	4 864	14 432	1 456
c7552	207	108	3 512	7 232	19 312	545

Table 1. ISCAS85 models (V and C denote the total number of variables and the number of clauses respectively)

where $m_{\text{err}}(c, \omega, \omega^*)$ is defined as:

$$m_{\text{err}}(c, \omega, \omega^*) = \begin{cases} 0, \text{ if } (\omega, c) = (\omega^*, c) \\ 1, \text{ otherwise} \end{cases} \tag{3}$$

In (2) and (4) the injected fault is denoted as ω^* and (ω, c) is the state of component c in diagnosis ω.

Although M_{ia} has been applied successfully to the industrial track of DXC'09, it has been considered as unsuitable in the synthetic track. The metric has to be "patched" when, for example, a diagnostic algorithm produces no diagnosis (in this case the diagnostic algorithm is assumed to have produced the "all healthy" diagnosis) and makes little intuitive sense with multiple faults (note that the DXC'09 industrial track contains single fault scenarios only). Despite that, we have computed M_{ia} for Safari and the results are shown in Table 2.

Instead of M_{ia} the DXC organizers have chosen to compute the utility metric M_{utl}. For notational completness we include the formula here, for explanation, cf. (Kurtoglu et al., 2009).

$$M_{\text{utl}} = \sum_{\omega \in \Omega} \frac{|\omega^*|}{[|\omega| + c(|\omega^* \cap \omega|, |\text{COMPS} - \omega|)]} \tag{4}$$

In (4) ω^* is the injected fault for which the diagnostic algorithm has produced a diagnosis ω, and $c(n, m)$ is defined as the expected number of trials needed to isolate n out of m faulty components. For $n < m$, $c(n, m) \approx nm/(n+1)$. Note that M_{utl} is computed for each scenario. The "per system" metrics M_{UTL} is M_{utl} averaged over all scenarios of a system.

Table 2 shows the results of all metrics as well as the number of scenarios for each circuit and the number of classification errors per scenario. It can be seen that M_{ia} depends on the number of diagnoses Safari produces and the latter depends on the parameter N in Alg. 1.

To summarize the experimental results, Safari has achieved good performance on both M_{ia} and M_{utl} while keeping the memory and CPU requirements (M_{mem} and M_{cpu}) very low. For comparison to the other diagnostic

sys	M_{ia} total	M_{UTL}	# of scenarios	M_{ia} per scenario	M_{cpu} [ms]	M_{mem} [Kb]
74182	145.6	0.4137	50	2.9	51	154
74L85	73.6	0.2433	28	2.6	68	223
74283	107.3	0.158	30	3.6	60	229
74181	258.9	0.1504	54	4.8	64	401
c432	253.4	0.0871	45	5.6	115	878
c499	2 326.4	0.0622	141	16.5	130	1 094
c880	3 897.8	0.0843	198	19.7	203	1 945
c1355	1 551.7	0.0295	98	15.8	296	2 759
c1908	2 302.8	0.0179	127	18.1	538	4 134
c2670	2 419.8	0.0647	168	14.4	937	5 867
c3540	373.3	0.0319	36	10.4	1 674	7 900
c5315	7 658.6	0.0165	248	30.9	3 091	11 316
c6288	2	0.0008	1	2	3 530	12 037
c7552	3 103	0.0317	176	17.6	11 817	16 679

Table 2. SAFARI DXC'09 results.

algorithms cf. (Kurtoglu et al., 2009). The low CPU and memory requirement is not a surprise considering the stochastic nature of SAFARI. Computing multiple-fault diagnoses close to the actually injected faults is practical with a cheap and simple stochastic algorithm.

5. CONCLUSION

This paper described the use of LYDIA and SAFARI for diagnosing the synthetic track system in DXC'09.

We have been satisfied with the results of SAFARI, we have learned a lot from DXC'09 and we would like to go further in building real MBD systems based on LYDIA and SAFARI.

We realize our results have been biased due to the fact that SAFARI had been used in creating the diagnostic scenarios (Kurtoglu et al., 2009) and the authors of this paper have been involved in designing the DXC challenges. In the future we plan to study the other algorithms competing at DXC'09, understanding the bias and removing it for future competitions.

REFERENCES

Brglez, F. and Fujiwara, H. (1985). A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *Proc. ISCAS'85*, 695–698.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7), 394–397.

De Kleer, J. (1986). Problem solving with the ATMS. *Artificial Intelligence*, 28(2), 197–224.

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

de Kleer, J. and Williams, B. (1987). Diagnosing multiple faults. *Artificial Intelligence*, 32(1), 97–130.

Feldman, A., Pietersma, J., and van Gemund, A. (2006). All roads lead to fault diagnosis: Model-based reasoning with LYDIA. In *Proc. BNAIC'06*.

Feldman, A., Provan, G., and van Gemund, A. (2008a). Computing observation vectors for max-fault min-cardinality diagnoses. In *Proc. AAAI'08*.

Feldman, A., Provan, G., and van Gemund, A. (2008b). Computing observation vectors for max-fault min-cardinality diagnoses. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence (AAAI'08), Chicago, Illinos, USA*, 911–918.

Forbus, K. and de Kleer, J. (1993). *Building Problem Solvers*. MIT Press.

Hansen, M., Yalcin, H., and Hayes, J. (1999). Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test*, 16(3), 72–80.

Kurtoglu, T., Narasimhan, S., Poll, S., Garcia, D., Kuhn, L., de Kleer, J., van Gemund, A., Provan, G., and Feldman, A. (2009). First international diagnosis competition - DXC'09. In *Proc. DX'09*.

McAllester, D. (1990). Truth maintenance. In *Proc. AAAI'90*, volume 2, 1109–1116.

Siddiqi, S. and Huang, J. (2007). Hierarchical diagnosis of multiple faults. In *Proc. IJCAI'07*, 581–586.