MODELING AND COMPILATION ASPECTS OF FAULT DIAGNOSIS COMPLEXITY

Jurryt Pietersma Alexander Feldman Arjan J.C. van Gemund
Delft University of Technology
P.O. Box 5031
Delft, Netherlands NL-2600 GA
+31 (0)15 278 1935
{j.pietersma, a.b.feldman, a.j.c.vangemund}@tudelft.nl

Abstract - Model-Based Diagnosis (MBD) is a promising approach for fast and accurate diagnosis of root cause of failure for complex systems. Critical success factors in MBD are accurate realism of the model and diagnostic inference speed. Both success factors are inherently at conflict. In this paper we demonstrate how to model system components that are typically found in technical systems in a qualitative approach, thereby limiting the need for more variables as well as increased domain resolution. Furthermore we outline the computational trajectory from model to diagnosis, in which we distinguish between (off-line) model compilation to an intermediate representation, and the subsequent (on-line) diagnosis step, based on actual observations. We use the fuel system of an Airbus 330 and a scenario based on a real-life incident as leading example. The results show that our framework is well suited for qualitative models and that these models provide a realistic representation and accurate diagnosis. From our results we conclude that the selected knowledge representation can speed-up the diagnostic process to up to two orders of magnitude.

INTRODUCTION

For complex systems, fast and accurate diagnosis of root cause of failure is a crucial aspect of operational safety and a critical factor in reducing system down-time. Model-Based Diagnosis (MBD) is a promising approach to reduce fault diagnosis time and cost. It is an automated fault diagnosis method based on first principles. With this method the system health state is inferred from a compositional, design-like system model and observations of system inputs and outputs. The model contains the relations between system inputs, health state, and partial observable behavior.

MBD has advantages over (automated) symptom-based methods as design changes only require equivalent changes in the model. In contrast, for symptom-based methods failure symptoms may become totally different, which would require major overhaul of the symptom to root cause mapping. Moreover, diagnosis models can be partly generated from existing engineering models.

Critical success factors in MBD are accurate realism of the model and diagnostic inference speed. Increasing model accuracy usually involves the use of more system variables (components, signals) and increasing the domain resolution of those variables, thereby increasing the size of the model. As inference complexity is typically exponential in the size of the model, both success factors are inherently at conflict.

In this paper we discuss both the modeling and diagnostic inference aspects in MBD. With respect to modeling, we demonstrate how to model system components that are typically found in technical systems in a qualitative approach, thereby limiting the need for more variables and increased domain resolution. We also address the modeling of dynamic behavior for these components.

This work has been carried out as part of the TANGRAM project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under grant TSIT2026.

With respect to inference, we outline the computational trajectory from model to diagnosis, in which we distinguish between (off-line) model compilation to an intermediate representation, and the subsequent (on-line) diagnosis step, based on actual observations. A speed-up of the diagnostic inference can be achieved by compiling the model into a knowledge representation that allows diagnostic queries to be answered in a time polynomial to the size of the representation. In this paper we address the following representations for Boolean and finite integer (FI) problems: Well-formed formula (Wff), Conjunctive Normal Form (CNF), Disjunctive Normal Form (DNF), Ordered Binary Decision Diagrams (OBDD), and semi-decomposable Negation Normal Form (sdNNF). Exploiting hierarchy is another potential source of speedup. This can be used in both compilation [2], [5] and run-time approaches [11].

The above issues are discussed in the context of an MBD implementation based on the modeling formalism LYDIA (Language for sYstems DIAgnosis) and associated toolset that have been developed specifically for this purpose [10]. The language core is propositional logic, enhanced with a number of syntactic extensions for ease of modeling. Currently, the toolset features a simulator and a number of diagnosis tools, based on satisfiability solving (SAT), conflict-directed search (CDA*) [13], and hierarchical A* [7], [8], [11] solving. Automatic model generation is currently supported for a subset of Verilog and Netlists.

Apart from a number of specific, small examples to feature specific aspects of our modeling approach and compilation techniques, the paper includes an elaborate case study, involving the Airbus 330 fuel system. The case study shows how the system is modeled and diagnosed in the case of the 2001 leakage incident as reported in [9] and shows how MBD could have prevented the pilots from taking the wrong actions.

The remainder of this paper is organized as follows. In Section II we demonstrate how to model systems with LYDIA. In Section III we discuss the Airbus 330 case study. In Section IV we present the knowledge representations. In Section V we measure the performance of these representations in a number of diagnostic experiments. Section VI follows with the overall conclusions from our work.

MODELING

We introduce the LYDIA language by modeling a state-less valve system. In this example we also develop a modeling method for time-dependent behavior which we apply to model systems *with* state.

Valve example

We model a valve as a component with an incoming and outgoing flow. For a healthy valve, the valve control variable determines the outgoing flow. A *true* control variable implies an open valve for which the outgoing flow is equal to the incoming, and a *false* control variable implies a closed valve for which the outgoing flow is zero, i.e., *false*,

$$control \rightarrow (flowOut = flowIn)$$
$$\neg control \rightarrow \neg flowOut$$

which corresponds to the following Lydia code,

```
flowOut = control ? flowIn : false;
```

In diagnosis the number of observations is typically limited. For this component, only the control variable and the outgoing flow are observable. Listing 1 shows the complete Lydia model in which the valve behavior is dependent on the health variable *h*.

The keyword **system** indicates the definition of a component. Health variables, which are to be

```
system Valve (bool h, flowIn, flowOut, control )
{
  // variable attributes
  attribute health (h) = true;
  attribute probability (h) = h ? 0.99 : 0.01;
  attribute observable (flowOut,control) = true;

  // valve model: control=true implies open valve
  if (h) {flowOut = control ? flowIn : false;}
}
```

Listing 1 LYDIA valve model.

solved by the diagnostic engine, are declared by the attribute **health**. The attribute **probability** declares an a priori probability which is used as a search heuristic for the diagnostic inference, as well as for the ultimate ranking of the inferred diagnoses, since observations usually admit multiple diagnoses, as shown in the sequel. The attribute **observable** assigns those variables that are observable.

Since *flowIn* is not observable the only exclusive fault that can be detected is that of a leaky valve. The observations for this fault are *control* = *false* and *flowOut* = *true* which is only consistent for *h* = *false*. For all other observations *h* = *false* and *h* = *true* are *both* consistent, which illustrates that limited observability typically leads to limited diagnosability (multiple/ambiguous diagnoses).

As in many systems, components share connections which can be sensed for better diagnostic reasoning. Consider a system of two identical, parallel valves as depicted in Figure 1. The LYDIA model is given in Listing 2.

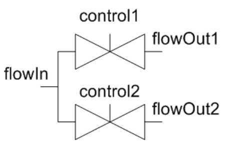

Figure 1 Two valves system.

```
#include "valve.sys"

system twoValves (
  bool h[1:2], flowIn, flowOut[1:2], control[1:2] )
{
  system Valve valve[1:2];

  forall (i in 1 .. 2) {
    valve[i] (h[i], flowIn, flowOut[i], control[i]);
  }
}
```

Listing 2 LYDIA model of two valves.

Both valves are fed with the same ingoing flow. If we observe that both valves are commanded open (*control1=control2=true*) while only valve two has outflow (*flowOut1 = false, flowOut2 = true*) we obtain the following diagnoses,

1. *h1=false h2=true*
2. *h1=true h2=false*
3. *h1=false h2=false*

Diagnosis 1 and 2 are equally probable, while diagnosis 3 (double fault) is much less probable. However, the second diagnosis does not represent expected physical behavior as *h1* = *true* implies *flowIn* = *flowOut1*= *false* which implies that the second valve would have a failure that spontaneously generates flow. To exclude this non-physical behavior from the model we need to extend it as follows,

```
if (not(flowIn)) {not(flowOut);}
```

Now the diagnosis reduces to either a single failure of valve 1 or a double failure of both valves, corresponding to the expected physical behavior of the (failed) system.

We can also improve the diagnosability of a system by expanding a model and observations in time. We revert back to the single (extended) valve model and assume that during a specific time horizon the input flow is constant. Let *flowOut*[*k*] denote the value of *flowOut* at the discrete time step *k*. If we observe *control* = *true*, *flowOut*[1] = *true*, *flowOut*[2] = *false* then the valve must be broken at *k* = 2.

Systems with state

We can also apply this temporal extension to systems with state. Consider a D-flip-flop as shown partially in Listing 3. For this system the output *o* at a certain time step *k* depends on the input *i* at *k* - 1. Consider the observations in Table 1. The diagnosis (last column) is correctly inferred because of the temporal relations between the observations.

A similar example is that of a robot arm. Due to space limitations we will only discuss the reasoning and not show the listing or the results. The arm holds a block, the position of which is measured by a sensor. A typical intermittent failure would be a sensor indicating the position of the block going from high to low and back to high again. If correctly modeled, the physical constraints ensure that a block does not spontaneously change position from low to high. This al-

```
system DFlipflop (bool s[0:1], r[0:1], i, o[0:1])
{
  ...

  sEvent=(!s[0] and s[1]);
  rEvent=(!r[0] and r[1]);

  if (h) {
    /* reset takes precedence */
    if (rEvent) {o[1]=false;}
    if (sEvent and !rEvent) {o[1]=i;}
    if (!sEvent and !rEvent) {(o[1]=o[0];}
  }
}

system DFlipflopTime ()
...
  forall (k in 1..5) {
    dff[k](s[k-1:k], r[k-1:k], i[k], o[k-1:k]);
...
```

Listing 3 LYDIA model of a D-flip-flop.

lows LYDIA to correctly infer that the sensor must be broken. Note that it is also possible to assign a health condition for the assumption of physical behavior, in this case gravity.

AIRBUS 330 CASE STUDY

As a modeling example we use the fuel system of the Airbus 330 aircraft. Figure 2 shows its schematics. The composition of the model is determined by the level of detail we need to obtain in a system diagnosis. We assume that identifying one or more of the field-replaceable components in Figure 2 is adequate for this purpose, therefore this schematic dictates the topology of the model.

Next, we need to determine the proper system variables to be modeled. For this, the main functionality of the system is used as guidance. The main function of the fuel system is to provide an uninterrupted supply of fuel to the engine. This supply should have a specific quantity and should be controllable by the pilot. Hence, we choose fuel mass and its derivative fuel mass flow, as leading system variables. These variables are influenced by all system components. This influence is partially controllable and measurable.

As example we discuss the engine component listed in Listing 4. In this listing we also see the use of **type** to define new variable types, **enum** to define variables with a FI domain, and **struct** to define structures. An engine is considered healthy if it is running when it is turned on (control variable) and there is nominal flow of fuel going into the engine. The side variable determines whether this engine is mounted on the left side (*false*) or on the right side (*true*). The mass flow direction *v* has to correspond to this.

The behavior of the other components is defined in a similar fashion. All components are used in a top-level system model. This is where the components are instantiated and the model topology is created by sharing the variables between components. For this particular case study it is not necessary to reason about the system across time,

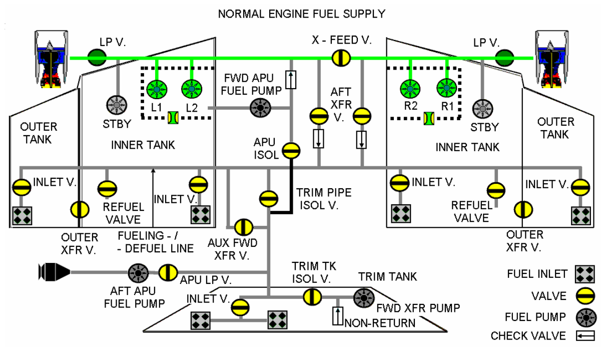

Figure 2 Schematics of the fuel system of the Airbus 330 [12].

thus for this model we do not use the temporal extensions as discussed earlier.
We consider the failure scenario as described in [1] and [9]. In this incident a leak near the right engine causes the following symptoms 1) a low oil temperature due to the increased flow rate through the right heat exchanger, and 2) a fuel imbalance between the left and right wing, initially compensated by a fuel flow from the trim tank. The pilots fail to combine these symptoms into a single root cause of failure. Instead they blame the problem on a broken oil temperature sensor and fail to take into account the fuel imbalance. As a result of this they did not close the cross feed valve thereby allowing more fuel to leak than necessary which aggravated the problem. Ultimately the fuel runs out causing a flame out in both engines. Fortunately the pilots manage to safely land the plane by gliding it to the runway.

Table 2 shows the different phases in the incident, the accompanying values of the relevant observations, and the best, i.e., computed as most probable, diagnosis of the relevant components as inferred by LYDIA. In total there are 51 observable variables and 46 components each with its own health variable. In the nominal flight condition there is a nominal fuel supply to the engines and the trim tank is not transferring any fuel. The best diagnosis is indeed the nominal diagnosis, which means that besides the right heat exchanger and engine all other components are also healthy.

```
/* mass type */
type Mass = enum {zero, low, nom, high};

/* one dimensional flow has size m and
direction v (false = -, true = +) */
type Flow = struct {Mass m, bool v};

/* Engine: model of engine.
Incoming nominal flow with engine switched on
implies running engine and vice versa.*/

system Engine (Flow flow,
  bool side, engineOn, engineRunning)
{
  ...

  if (h) {if (engineOn) {
    engineRunning =
      ((flow.v=side) and (flow.m = Mass.nom));
  }}
}
```

Listing 4 LYDIA model of the Airbus 330 engine.

The leak manifests itself at the other phases. Initially the fuel loss is compensated by an automatic fuel transfer from the trim tank, this is shown in the second value column LOW+AUTO. In this particular phase the flow from the trim tank compensates the higher outflow of the right wing tank, hence masking the higher fuel consumption due to the leakage. Only the low temperature of the oil in the right heat exchanger indicates a problem. At this phase, the diagnosis agrees to the diagnosis as inferred by the pilot, i.e., a failure of the heat exchanger temperature sensor. However as soon as the trim tank is unable to compensate, the fuel flow from tank increases to high (third value column, LOW+AUTO+LOSS). This affects the diagnosis, as the combination of an increased flow fuel rate and low oil temperature can not be caused by a single failure of the heat exchanger. Instead, the right engine is diagnosed to be the only single, and hence most likely, root cause of failure. The fourth column takes into the account the pilot action of opening the cross feed valve, which neither mitigates the problem nor affects the diagnosis.

Similarly to the valve example the correct diagnosis is inferred by combining multiple observations of interlinked system components. Though the flow through the fuel lines is not directly available as observation, LYDIA correctly reasons that a high flow rate out of the right tank and a high flow rate through the right heat exchanger can be consistently combined into a single root cause. LYDIA correctly identifies this root problem as a single failure in the right engine section, i.e., a *false* value for *REngine* corresponding to *h* in Listing 4.

Table 2 Incident phases, conditions, relevant observations, and best diagnosis for the Airbus 330 case study.

observation	**phase**			
	NOM	LOW AUTO	LOW AUTO LOSS	LOW AUTO LOSS X
LInletV	false	true	true	true
RInletV	false	true	true	true
fRTank.m	nom	nom	high	high
xFeedV	false	false	false	true
ROil	nom	low	low	low
fTrim.m	zero	nom	nom	nom
trimPump	false	true	true	true
trimIsolV	false	true	true	true
diagnosis				
RHeatExch	true	**false**	true	true
REngine	true	true	**false**	**false**
other	true	true	true	true

condtion	**description**
NOM	nominal flight conditions
LOW	low oil temperature
AUTO	automatic forward transfer, fuel transfer from trim tank to wing
LOSS	fuel loss in right wing
X	cross feed open

COMPILATION

In this section we discuss the process of computing diagnosis by using the LYDIA framework. For brevity we omit the diagnosis formalism [4] and sketch through some of the computational aspects of compilation. The illustration of this process is done with the Lydia toolkit, which contains a compiler, a number of preprocessing and transformation tools and several diagnostic engines. Due to space constraints these algorithms are not discussed here.

In order to perform diagnosis on a LYDIA model it is enough to translate it to a well-formed propositional formula (Wff) and to use an entailment mechanism for finding those diagnoses which are consistent with the system description and the observations, i.e., the diagnoses that explain the observations in terms of the health state of that system. The language is designed in such a way as to facilitate a conversion to a propositional Wff in polynomial time. This includes the normal language parsing, type-checking, expanding arrays and array quantifiers and processing of variable attributes.

The LYDIA language supports variables both in the Boolean and FI domains. The toolkit supports two approaches for unifying this – encoding FI variables as Booleans and vice-versa [6]. Encoding FI into Boolean is trivial and we will not discuss it for brevity. Working directly in the FI domain is a preferred option as it offers speed-ups of up to two orders of magnitude [6]. This approach has also been taken in the Airbus case study.

Up until now, we have discussed the compilation of the original model to a Boolean or FI Wff. As this is the conjunction of each system's Wff the actual hierarchy is preserved in this representation. This introduces the notion of hierarchical system, which will allow us to perform faster reasoning compared to algorithms working on "flat" representations[1] [7]. A hierarchical system description is a break-up of the system description into small partial system descriptions that are organized in a hierarchical structure with one system description at the highest hierarchy level. Similar to the hierarchical Wff representations, the introduction of hierarchical system descriptions allows us to define hierarchical CNF and hierarchical DNF (the latter is not DNF anymore but is a restricted form of Negation Normal Form). We call this hierarchical DNF semi-decomposable Negation Normal Form (sdNNF). A hierarchical system is simply a conjunction of Boolean or FI Wff. It is possible to discard this information (i.e., to flatten out the hierarchy) and to continue transforming the Wff in its flat representation.

Instead of completely flattening a hierarchical system as would be the case in traditional diagnosis, we selectively apply compilations on subsystems of system descriptions. The need to exploit hierarchy stems from the inherent high-computational price of MBD. By exploiting the hierarchical information and selectively compiling parts of the model it is possible to increase the diagnostic performance and to trade cheaper preprocessing time for faster run-time reasoning. Our hierarchical algorithm, being sound and complete, allows large models to be diagnosed, where compile-time investment directly translates to run-time speedup. Furthermore LYDIA repartitions and coarsens the original model in an attempt to minimize the subsystem connectivity, a process which leads to faster run-time fault diagnosis at the price of some pre-processing time. The algorithms implementing this are an important part of the reasoning tool-kit.

The reasons for the different compilation routes are three-fold. First, we need to reach logically equivalent representation which allows us to cross-validate the correctness of the tools. The implementation of other state-of-the-art techniques (like CDA* [13]) allows us to verify the final diagnostic result and to compare the diagnostic performance under fair conditions. Finally, and most importantly, almost any translation causes combinatorial explosion with some models. Models or sub models, depending on different characteristics (e.g., implicit or explicit fault modeling, etc.) can produce very different compilation results. More precisely, there are Boolean functions which have linear OBDD representation, but exponential irreducible CNF. Therefore it is beneficial to choose different representations for different models and parts of models. Full complexity analysis for all representations is impossible due to lack of space in this paper but is available in [3].

EXPERIMENTS

We demonstrate the effect of the different knowledge representations experimentally by diagnosing the example models discussed in the previous sections. Table 3 shows the resulting diagnosis time in milli-seconds. The model's number of health variables is represented by N. The diagnoses are inferred for a nominal and a failure scenario corresponding to the failures that were discussed earlier. In general we can conclude that for small example models the sdNNF is faster, with the D-flip-flop as notable exception. This speed-up comes at an investment in compilation time and size. For the large Airbus example we were unable to compile the sdNNF representations. The most dramatic speed-up is provided by FI representation of the Airbus model. In fact, the Boolean representation does not yield a diagnosis on a reasonable time scale.

The effect of the hierarchical preprocessing step on the diagnosis time is shown in Table 4. The three representations on which we perform diagnostic search are constructed as follows. The sdNNF/H models are the original high-level ISCAS-85 models converted to sdNNF (with tree

[1] For brevity, we refer to the classical diagnosis approach as "flat", i.e., non-hierarchical.

Table 3 Diagnosis performance for CNF (F) and sdNNF (H) representations of the example models.

model	N	search time [ms]			
		nominal		failure	
		F	H	F	H
valve	1	0.55	0.29	0.23	0.10
2Valves	2	0.98	0.59	0.64	0.26
valveExt	1	0.47	0.28	0.22	0.10
2ValveExt	2	0.98	0.58	0.64	0.26
valveT	2	0.99	0.60	0.68	0.35
valveTExt	2	0.98	0.62	0.68	0.27
D-flip-flop	5	2.54	15.72	2.37	10.77
robotArm	12	1.19	0.43	3.29	0.43
A330 Bool	46	44930	-	-	-
A330 Fl	46	200	-	390	-

depth *d*). By partially flattening to a depth *d*-1 we obtain sdNNF/P1 and by flattening to *d*-2 we obtain sdNNF/P2. By partial, compile-time flattening to sdNNF/P1, we gain run-time speed-up by a factor varying from 2 to 5.6 and with the deeper flattening to sdNNF/P2 the speed improvement varies from 2.9 to 12.3. For this improvement in speed we pay the price of increasing the representation size in comparison to the original sdNNF/H. The speed of the run-time search is determined by the depth and the quality of partial flattening, which depends on the original hierarchy. The sdNNF/H representations of the ISCAS-85 circuits, however, are obtained by sequential splitting of large nodes to prevent explosion in the size of the partially flattened sdNNF. This naive partitioning is the reason for the limited speedup in diagnosis time.

CONCLUSIONS

In this paper we have discussed how to qualitatively models systems in LYDIA for the purpose of MBD and how the complexity of these models affects the knowledge representations and the diagnostic inference time. Our results show that LYDIA provides a good framework for qualitative models and that these models provide a realistic representation and accurate diagnosis. This also holds for dynamic systems and in real world scenarios such as the Airbus 330 incident. From our results we conclude that the selected knowledge representation has a major influence on the diagnostic inference speed-up. Automatic selection of the optimal representation is subject of ongoing research.

Table 4 Diagnosis time of sdNNF representations for the benchmark suite models.

circuit	N	diagnosis time [ms]		
		sdNNF/H	sdNNF/P1	sdNNF/p2
74182	19	0.56	0.28	0.15
74283	40	1.41	0.40	0.30
74L85	41	1.28	0.65	0.45
74181	62	17.97	3.43	1.46
c432	146	61.27	10.93	10.00
c499	202	12.22	3.55	3.84
c1908	252	20.61	7.17	4.75
c880	383	38.44	12.43	4.51
c1355	514	102.34	40.04	16.36
c2670	983	981.73	333.20	111.47
c3540	1297	871.21	413.96	187.32
c5315	2202	23172.46	6786.24	2796.49
c6288	2416	1453.99	450.89	123.38
c7552	3024	4264.19	1533.95	746.96

REFERENCES

[1] http://www.rvs.unibielefeld.de/publications/compendium/incidents_and_accidents/airtransat236.html.
[2] Adnan Darwiche, "Model-based diagnosis using structured system descriptions," JAIR, vol. 8, 1998, pp. 165–222.
[3] Adnan Darwiche and Pierre Marquis, "A knowledge compilation map," Journal of Artificial Intelligence Research, vol. 17, 2002, pp. 229–264.
[4] Johan de Kleer, A. K. Mackworth and R. Reiter, "Characterizing diagnoses and systems," Artificial Intelligence, vol. 56, 1992, pp. 197– 222.
[5] Yousri El Fattah and Rina Dechter, "Diagnosing tree-decomposable circuits," in IJCAI'95, 1995, pp. 1742–1749.
[6] A. Feldman, J. Pietersma and A.J.C. van Gemund, "A multi-valued SAT-based algorithm for faster model-based diagnosis," in Proc. DX-06, June 2006.
[7] A. Feldman and A.J.C. van Gemund, "A two-step hierarchical algorithm for model-based diagnosis," in Proc. AAAI'06.
[8] A. Feldman, A.J.C van Gemund and A. Bos, "A hybrid approach to hierarchical fault diagnosis," in Proc. DX-05, June 2005, pp. 101–106.
[9] Government of Portugal Gabinete de Prevenção e Investigação de Acidentes com Aeronaves, "Accident investigation final report: Airbus 330-243 available at www.gpiaa-portugal-report.com," October 2004.
[10] http://www.fdir.org/lydia/
[11] Gregory Provan, "Hierarchical model-based diagnosis," in Proc. DX'01, 2001.
[12] "Red triangle productions airbus a330." http://www.redtriangle.com.
[13] Brian Williams and Robert Ragno, "Conflict-directed A* and its role in model-based embedded systems," Journal of Discrete Applied Mathematics, 2004.