Model-Based Diagnosis under Real-World Constraints*

Adnan Darwiche
Computer Science Department
University of California
Los Angeles, Ca 90095
darwiche@cs.ucla.edu

Abstract

I report in this paper on my experience over the past few years in introducing automated, model-based diagnostic technologies into industrial settings. In particular, I discuss the competition that this technology has been receiving from hand-crafted, rule-based diagnostic systems which has set some very high standards that must be met by model-based systems before they could be viewed as viable alternatives. The battle between model-based and rule-based approaches to diagnosis has been over in the academic literature for many years now, but the situation is different in industry where rule-based systems are dominant and appear to be quite attractive given the considerations of efficiency, embeddability and cost-effectiveness. My goal in this paper is to provide a perspective on this competition and to discuss a diagnostic tool, called DTOOL/CNETS, that I have been developing over the years as I tried to address the major challenges posed by rule-based systems. In particular, I will discuss three major features of the developed tool which were either adopted, designed or innovated to address these challenges: Its compositional modeling approach, its structure-based computational approach, and its ability to synthesize embeddable diagnostic systems for a variety of software and hardware platforms.

Keywords: model-based diagnosis, structure-based reasoning, rule-based systems, compiling knowledge, embedded reasoning systems, compositional modeling.

1 Introduction

System diagnostics, a classical engineering concern, has become more important than ever given the increasingly complex systems now central to many industrial, and non-industrial, operations.

Traditionally, industrial diagnostic systems have been hand-crafted to reflect the knowledge of a domain expert. They take the form of if-then rules that associate certain forms of abnormal system behavior with faults that could have caused that behavior. Although an improvement over manual diagnostics, the dissatisfaction of industries with such hand-crafted systems has been growing for a number of reasons. First, the development of such systems has proven to be extremely difficult and time consuming. Second, it is almost impossible to provide formal guarantees on the quality of hand-crafted diagnostics given the ad-hoc manner in which they are typically developed. Finally, the maintenance of such systems has proven to be almost infeasible in light of changes to the underlying system. A little, but subtle, change to the system configuration or architecture may invalidate a large number of diagnostic rules and may qualify many others. It has proven quite difficult to isolate the invalidated rules, let alone updating them, under such changes.[1]

In an attempt to address these difficulties, I have been advocating the use of model-based diagnostic technologies by developing the DTOOL/CNETS system, which I aim to describe in this paper. A model-based diagnostic approach works simply as follows [Hamscher *et al.* 1992, Reiter 1987, Forbus and de Kleer 1993, de Kleer and Williams 1987]. One develops a forward (simulation) model of the system to be diagnosed and then invokes diagnostic algorithms on this model to decide whether a certain system behavior is normal

*To appear in AI Magazine, 2000.

[1]These observations, and related ones throughout the paper, are based on years of collaborative work with engineers in several industries, including aerospace, industrial automation and commercial avionics.

or abnormal. In case the behavior is abnormal, further algorithms can be invoked to isolate the system components responsible for the abnormality.

Model-based approaches to diagnostics overcome the main difficulties with hand-crafted approaches. First, developing a forward system model is far easier than identifying diagnostic rules (assuming that diagnostic rules are identified by reflecting on the system behavior, as opposed to using automated methods such as fault insertion and simulation). Second, algorithms do exist for answering diagnostic queries based on a system model and these algorithms are known to be sound and complete (but many practical implementations of these algorithms compromise completeness or soundness to reduce computational complexity). Third, model-based approaches do solve the diagnostics-update problem: If the system configuration or architecture changes, and such changes are reflected in the model, system diagnostics are automatically updated. There is no need to change the diagnostic algorithms as long as the system model is kept up-to-date.

Despite these "attractions," I have found out that users of hand-crafted, rule-based diagnostic systems are quite reluctant to replace these systems with model-based ones. It is my current understanding that such reluctance is due to the following merits of hand-crafted, rule-based diagnostic systems which are typically not a mark of standard model-based approaches:

Rule-based systems are quite intuitive. Engineers and high-level decision makers understand if-then rules; they also find it easy to comprehend what a rule evaluator is. On the contrary, model-based approaches require the mastering of a modeling language and their associated algorithms seem to some engineers as performing "magic."

Rule-based diagnostics is relatively straightforward to embed into systems; all that is needed to implement is a rule evaluator which is known to require very little software and hardware resources. Model-based approaches, however, are quite complex, requiring significant software and hardware resources.

Rule-based diagnostics raise no efficiency issues as the evaluation of diagnostic rules can typically be accomplished in linear time (in the size of rules) in the worst-case. On the contrary, many model-based approaches are known to be practically infeasible given their memory and time demands.

Rule-based systems can be built incrementally — in principle, such systems can be based on a handful of rules! It is perceived, however, that a model-based system would require a sophisticated and detailed model before it can be deployed.

It is for these reasons that industries are reluctant to trade-in a rule-based system for a standard model-based system. From an engineering viewpoint, the above merits of rule-based systems contribute to the bottom-line of cost-effective system engineering and are not to be given up easily, even for guarantees of soundness, completeness, and automatic update which are the promises of model-based approaches.

The development of the DTOOL/CNETS *system has been driven by an interest to resolve this dilemma — that is, to offer the benefits of model-based diagnostics while not compromising the merits that have made rule-based diagnostics the dominant solution in industry.*

DTOOL/CNETS is composed of two main modules: A compositional modeling module, known as DTOOL, and a computational engine based on abstract causal networks, known as CNETS [Darwiche 1992]. The main goal of this paper is to present an overview of the innovations and design decisions that I have employed in DTOOL/CNETS in order to address the above dilemma. In particular, I will start by presenting a simple diagnostic problem and then discussing how rule-based and model-based approaches would address that problem. This provides an opportunity to contrast the two approaches and to identify the obstacles that model-based systems must overcome in order to be competitive with their rule-based counterparts. I then follow by discussing the innovations and design decisions that I have employed in DTOOL/CNETS to overcome these obstacles. Specifically:

I discuss in Section "A Three-Layer Modeling Approach" how to shield system engineers from the difficulties typically associated with building system models. As we shall see, only the very last layer in this approach is exposed to average system engineers where they only see domain-specific components from which they can build system models. The underlying second level consists of block diagrams, a traditional engineering tool for building system models. System engineers who are technologically aware may want to address this level, but it is otherwise reserved for AI knowledge engineers working

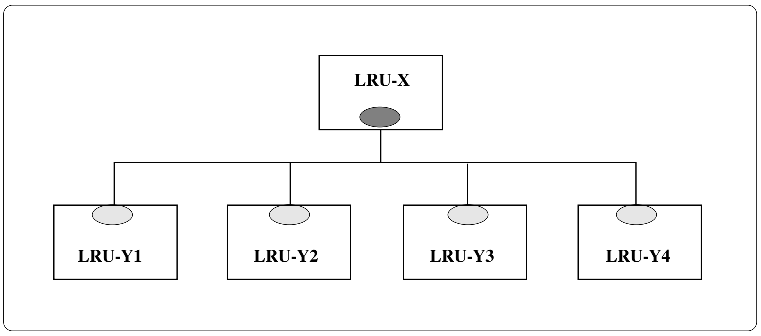

Figure 1: A portion of an avionics system.

with domain experts. The third level is the causal-network level on which model-based diagnostic algorithms operate and it is typically reserved for CNETS developers.

I then dedicate Section "Embeddable Diagnostics" to an innovative compilation technique employed by CNETS to address the embeddability challenge posed by rule-based systems. This technique allows CNETS to compile a system model into an embeddable diagnostic system which can require modest software and hardware resources to operate. Moreover, the compilation technique can easily generate code for multiple software and hardware platforms, rendering the deployability of model-based diagnostic systems an economical reality.

I finally dedicate Section "The Structure-Based Guarantees" to the computational performance of DTOOL/CNETS. I show in this section that CNETS algorithms are structure-based, which means that the guarantees CNETS offers on the time to generate and the space to store an embeddable diagnostic system are a function of the system structure, its connectivity in particular. The less connected the system is, the easier it is to diagnose and embed. Therefore, the performance of CNETS is tied to a very meaningful engineering measure, which provides engineers with a certain level of control through their influence on the system design process.

2 Deriving System Diagnostics: The Problem

I start in this section by laying out a simple diagnostic problem and discussing how it is typically addressed by a hand-crafted, rule-based approach versus an automated, model-based one. This allows us to explain in more details the trade-offs between the two approaches and the challenges that automated approaches tend to face when they are introduced as alternatives.

Consider the system in Figure 1 which depicts a number of LRUs (line replaceable units) in an avionics system. A transmitter on LRU-X puts data on the wire connecting it to LRU-Y1, . . . , LRU-Y4. This data is collected by the corresponding LRUs using receivers. There is a monitor inside each LRU which detects whether the corresponding receiver is seeing any data. The problem is as follows: Given a present/absent/unknown report from each LRU, determine whether the transmitter or any of the receivers are faulty.

2.1 Rule-based diagnostics

To hand-craft a rule-based diagnostic system for this problem, one would proceed by identifying combinations of reports that would indicate a failure and then capture them into diagnostic rules. For example, suppose that LRU-Y1 sends an absent-report, and each of the other three LRUs sends an unknown-report. That is, we know that no data is being received at LRU-Y1, and it is not known whether data is being received at the other LRUs. In such a case, we are not sure whether the transmitter is faulty (no data is being transmitted) or whether the Y1-receiver is faulty (data is being transmitted, but the receiver is not seeing it). This leads to the following diagnostic rule:

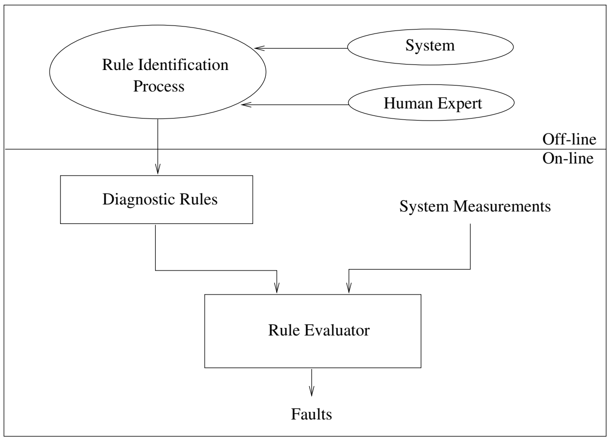

Figure 2: The process of hand-crafting a rule-based diagnostic system.

```
If R-Y1-Out=Absent,
   R-Y2-Out=Unknown,
   R-Y3-Out=Unknown,
   R-Y4-Out=Unknown,
Then either the transmitter of LRU-X is faulty or the receiver of LRU-Y1 is faulty.
```

Here, `R-Y1-Out` represents the data status at receiver `R-Y1`.

Suppose, however, that LRU-Y1 sends an absent-report, while each of the other LRUs sends a present-report. In this case, the receiver of LRU-Y1 is to be blamed. A rule would then be written to this effect:

```
If R-Y1-Out=Absent,
   R-Y2-Out=Present,
   R-Y3-Out=Present,
   R-Y4-Out=Present,
Then the receiver of LRU-Y1 is faulty.
```

The crafting of a rule-based diagnostic system is then a process of identifying such rules through a reflection process which attempts to explicate abnormal system behaviors and the causes for these behaviors. The process is sketched in Figure 2 which identifies two phases: an off-line phase leading to a rule database, and an on-line phase which is typically embedded.

The following table summarizes the pros and cons of rule-based approaches before I move on to describing the model-based approach to this problem.

Pros:
- the rules are quite intuitive; - they can be embedded in a system cost-effectively; - executing diagnostic rules can be done quite efficiently.
Cons:
- the process of rule identification can be error-prone; - the quality of identified rules is a reflection of the expert's trouble-shooting abilities; - the rules can be conflicting, requiring subtle resolution techniques; - it is not clear how to test whether the identified rules are correct; - it is not clear how to test whether all applicable rules have been identified.

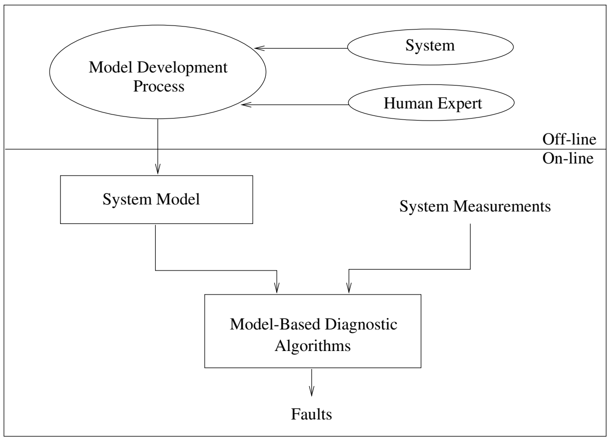

Figure 3: The process of automatically generating system diagnostics using a model-based approach.

2.2 Model-based diagnostics

A model-based approach to this problem would proceed as follows. We first develop a model of the system (shown in Figure 1). Then, given a combination of LRU-reports, we pass the model and reports to an algorithm which comes back with the components to blame (if any).

There are many realizations of model-based diagnostics, each based on a different modeling language and, hence, a different set of algorithms. The classical framework, however, is the one described in [Reiter 1987, de Kleer *et al.* 1992] where a system is modeled using symbolic logic. If we were to model the above system using logic, we would have something like this:

```
;;; A transmitter generates data iff it is healthy
If T-X-Mode=Ok, then T-X-Out=Present
If T-X-Mode=Faulty, then T-X-Out=Absent

;;; A receiver detects the transmitted data if it is healthy
If R-Y1-Mode=Ok and R-Y1-In=Present, then R-Y1-Out=Present
If R-Y1-Mode=Ok and R-Y1-In=Absent, then R-Y1-Out=Absent

;;; The transmitter output is connected to the receiver input
T-X-Out=R-Y1-In

;;; Symmetric models for LRU-Y2, LRU-Y3 and LRU-Y4.
.
.
.
```

Given an appropriate suite of diagnosis algorithms [Hamscher *et al.* 1992, de Kleer 1986, Reiter and de Kleer 1987, Darwiche 1998b], this model is all we need to identify faulty components given a set of LRU reports. For example, if the reports were such that

```
R-Y1-Out=Absent
R-Y2-Out=Present
R-Y3-Out=Present
R-Y4-Out=Present
```

then the diagnostic algorithms would conclude that the Y1-receiver is faulty.

The notion of a *candidate diagnosis* is central to the theory of model-based diagnosis. A candidate diagnosis is a system state of health, represented by an assignment of a health mode to each system component.

In this example, we have five components, `LRU-X`, `LRU-Y1`, `LRU-Y2`, `LRU-Y3`, `LRU-Y4`. Each of these components can be either faulty or ok, therefore leading to $2^5 = 32$ diagnosis candidates. Given a system model Δ and some observed behavior α, model–based diagnosis concerns itself largely with candidates consistent with $\Delta \wedge \alpha$, which are called *consistency–based diagnoses.* Typically, we have many consistency–based diagnoses, some of which are more plausible than others. One of the most common methods for measuring plausibility is the *minimum-cardinality* criterion. That is, a diagnosis is more plausible than another precisely when it involves a smaller number of faults. Therefore, a key computational task in model-based diagnosis is the generation of minimum-cardinality, consistency-based diagnoses. The example I provided above, and the DTOOL/CNETS system, is based on such an approach. An important issue in model–based diagnosis is whether to allow multiple faults, or simply assume the existence of a single fault. Under a single-fault assumption, we only have a linear number of candidate diagnoses (instead of exponential). This simplifies the computational problem considerably. Model-based approaches typically admit multiple faults. Rule-based approaches typically allow single faults only.

The process of developing a model-based diagnostic system is sketched in Figure 3, which splits the process into two phases: the off-line phase which involves model development and the on-line phase which is comprised of a system model and a suite of algorithms. A number of observations are in order about this approach. First, the system model shown above is expressed using if-then rules, but these rules are quite different from diagnostic rules:

They are local: each rule references only one component.

They are forward: each rule specifies the output of a component in terms of its inputs and health.

Diagnostic rules, however, are global and inverted — they typically reference more than component and they map inputs and outputs to components' health.

Second, as far as the pros and cons of this model-based approach, we have the following:[2]

Pros:
- it guarantees correct and complete answers to diagnostic queries; - system diagnostics are updated automatically as the model is updated; - developing a forward system model is far easier than developing diagnostic rules.
Cons:
- developing a system model at the right level of detail is quite a challenge; - the on-line portion of the architecture is quite demanding as far as software and hardware resources are concerned, especially when compared to the on-line portion of a rule-based approach; - no algorithm exists which can operate efficiently on all possible system models — in fact, most of the known algorithms operate quite inefficiently on many system models.

As mentioned earlier, the design and development of DTOOL/CNETS was driven by the desire to attain the benefits of model-based diagnostics, while addressing their drawbacks to the extent possible. This desire has motivated and inspired the following innovations and design decisions:

A three-layer modeling approach, which allows system engineers with different knowledge levels to model systems at different levels of abstraction.

An approach for compiling system models, which offers model-based diagnostics in a cost-effective, cross-platform embeddable form.

A structure-based computational approach which is known to be efficient for a useful class of systems.

These features are explained in detail in the following three sections.

[2]The completeness of model-based diagnosis algorithms is with respect to the given model. That is, any diagnostic conclusion which follows from the model and laws of logic (or probability, if a probabilistic approach is used) are guaranteed to be discovered by these algorithms. But such conclusions can be intuitively incomplete—that is, they may not be as strong as our intuition would suggest—if the model itself is incomplete. One of the tenets of model-based diagnosis though is that ensuring the completeness of a forward, simulation model is easier than ensuring the completeness of a set of diagnostic rules. This argument has some technical merits: there are many complete models, which have linear size, yet entail an exponentially-sized, complete set of diagnostic rules.

3 A Three-Layer Modeling Approach

Both rule-based and model-based diagnostic approaches require input from a domain expert. In the rule-based approach, the expert must provide diagnostic rules; in the model-based approach, the expert must provide a forward system model. As I have illustrated in the previous section, diagnostic rules are global and inverted: they typically reference multiple components and they map a system behavior into a system's state of health. A forward system model, however, is composed of local, forward rules which appear to be more intuitive than global, inverted ones.[3] Despite this apparent intuitive appeal, I have found that system engineers are quite reluctant to pay the cost of specifying a system model for obtaining the benefits of model-based diagnostics. I have tracked this reluctance to two main reasons:

Modeling-for-diagnosis problem: Engineers are accustomed to building simulation models which are typically quite detailed and can be very demanding if the system to be modeled is complex enough. And it is this demand that causes engineers to shy away from committing to develop the models necessary for diagnosis as they expect these models to resemble the ones they typically build for simulation purposes. It has been my experience, however, that detailed simulation models are not necessarily required by model-based diagnosis. Instead, the type of models required by model-based diagnosis depend mainly on the type of system failures one is interested in covering. Consider the system in Figure 1 for an example. If we are only interested in isolating failures of transmitters and receivers, then these components (and their interconnections) are all we need to model — there is no need to model the details of each LRU. Therefore, although a system may be quite complex, its model-for-diagnosis may be quite simple. Deciding the scope and granularity of a diagnosis model, however, is quite challenging and typically requires extensive experience in applying model-based diagnostic techniques. Average system engineers do not seem to have this kind of expertise and, therefore, are typically not in a position to provide the models required by model-based diagnostic approaches.

Modeling-language problem: Each model-based diagnostic approach is based on some formal language which is used to specify the system models of interest. The most common languages in use are based on either probability, logic or a combination. It is on these "native" languages that model-based algorithms operate on and it is quite common for AI researchers to expect engineers to specify their models directly using these languages. DTOOL/CNETS adopts causal networks as the modeling language, which support both probabilistic and logical modeling [Darwiche 1992, Darwiche and Pearl 1994, Darwiche 1998b] — see Figure 6 for an example causal network. It was my initial expectation that engineers would model their systems by building such causal networks, which clearly required a certain level of understanding on the use of causal networks — and any adopted modeling language for that purpose. I have found out, however, system engineers typically lack the background or interest necessary to develop this prerequisite level of understanding.

The choice of causal networks as a modeling language is driven by a number of technical considerations, some of which will be detailed later. But whether causal networks or any other specialized modeling language were to be used, the modeling-language problem demanded that average system engineers not be exposed to it. My solution to this problem has been to provide engineers with a modeling language that they are familiar with and then translate models specified using that language into causal network models. This solution is detailed in the following section. The solution to the modeling-for-diagnosis problem is then discussed next.

3.1 Block diagrams

I have chosen block diagrams to address the modeling-language problem. This is a classical modeling tool that most engineers are familiar with. Each system component is modeled as a block which has inputs, outputs and a health mode. Once these component blocks are defined, modeling a system becomes a matter of creating and connecting blocks together.

[3] I am assuming a relatively new system about which we do not have much experience. If a system has been in service for a long time, some technicians tend to develop good understandings of how certain behaviors associate with certain faults. In this case, diagnostic rules are not being developed by reflecting on the behavior of a newly designed system, but are being recalled from previous experience.

```
(define-block-class receiver
   (menu-name "Receiver")
   (shape ... )
   (outputs ((o (title "o")
                (shape ...)
                (values (present absent)))))

   (inputs
     ((p (title "p")
         (shape ...)
         (values (true false)))
      (i
         (title "i")
         (shape ...)
         (values (present absent)))))

   (modes ((ok 0) (faulty 1)))

   (equation (
        ;;;; if mode is ok and power is on, then output = input
        (:if (:and (= mode ok) (= p true) (= i present))
           (= o present))
        (:if (:and (= mode ok) (= p true) (= i absent))
           (= o absent))
        ;;;; if mode is faulty or power is off,
        ;;;; then no data on output
        (:if (:or (= p false) (= mode faulty))
           (= o absent)))
 )
```

Figure 4: A block definition for modeling a data-receiver component.

Figure 4 depicts the definition of a block which represents a data-receiver component. The block has two inputs, one representing transmitted data and the other representing power. It has only one output representing received data. The block definition indicates two health modes for this component: ok and faulty. The numbers associated with these modes represent order-of-magnitude probabilities estimating the likelihood of different health modes [Darwiche and Goldszmidt 1994].

Once a block is defined, DTOOL makes it available to the engineer as part of a component library. To model a system, the user engages in a process of creating instances of components (selected from the library) and then connecting the inputs and outputs of these components. Figure 5 depicts a screen shot from DTOOL showing a system model that uses the receiver block defined in Figure 4, in addition to a transmitter block the details of which are omitted here.

DTOOL can translate block-diagram models into causal-network models on which causal-network algorithms will operate. Figure 6 depicts the causal network which results from translating the block diagram in Figure 5. The translation is compositional: Each block representing a component is translated into a causal network segment. When blocks are connected, these causal network segments are connected together to form a global causal network. This compositional translation is made possible by an important property of causal networks: If the behavioral equations associated with each causal network segment are locally consistent, any causal network composed from such segments is guaranteed to be globally consistent. Therefore, to ensure the consistency of block-diagram models, DTOOL needs to only ensure the local consistency of individual blocks, a task that can be easily accomplished using a brute-force approach given the constant size of blocks. The formal definitions of local and global consistency of causal network models are given in [Darwiche and Pearl 1994, Darwiche 1998b]. In particular, a block-definition in DTOOL is intended to implement the notion of a component description as defined in [Darwiche 1998b], which formally states the conditions it must satisfy before it can be safely plugged into a global system model.

The block–diagram approach for modeling systems is very closely related to a number of modeling languages that appeared in the context of model–based reasoning [de Kleer and Brown 1984, Davis 1984, Genesereth 1984, Kuipers 1994, Franke 1989, Franke and Dvorak 1990]. For example, the CC language of

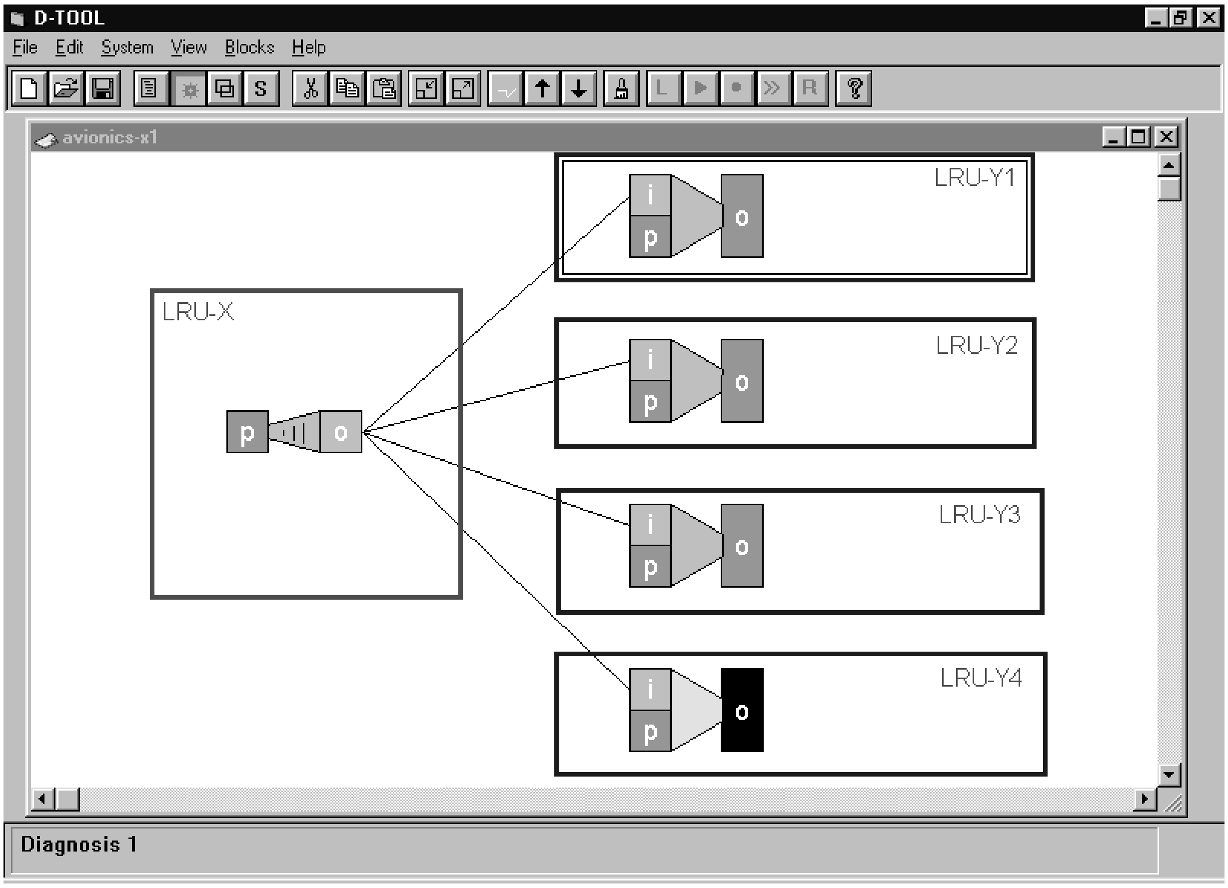

Figure 5: A DTOOL screen-shot depicting a system model specified using a block diagram. The figure shows a scenario where the user has set the reports of receivers Y1, Y2 and Y3 to present and the report of receiver Y4 to absent. DTOOL replies by indicating the existence of a single diagnosis and highlights receiver Y4 as the faulty component in the identified diagnosis.

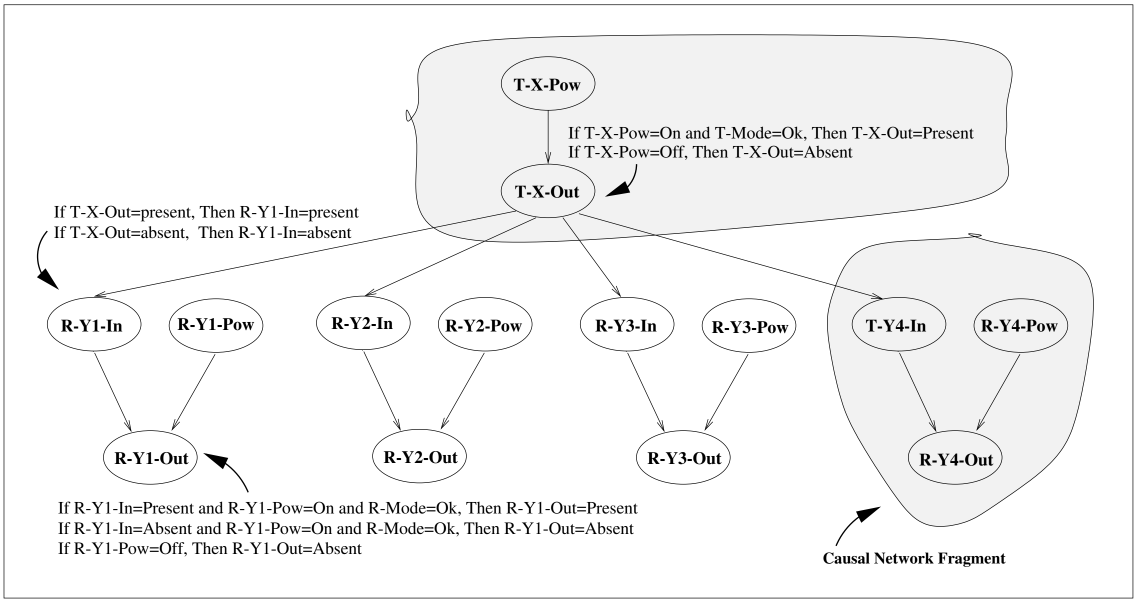

Figure 6: A *symbolic causal network* generated automatically for the system model in Figure 5—some of the Boolean equations are omitted for clarity of exposition. Symbolic causal networks are formally defined in [Darwiche and Pearl 1994] and their application to diagnostic reasoning is studied in [Darwiche 1998b] where they are called *structured system descriptions,* following the tradition on model-based diagnosis. In a nutshell, a structured system description has two parts: a system structure representing component interconnectivity and a set of Boolean equations describing the behavior of system components. The two parts must satisfy some coherence relations that are outside the scope of this paper.

[Franke 1989, Franke and Dvorak 1990] allows users to model systems using *components* and *connections* and then compiles such models into QSIM QDE models [Kuipers 1994]. I note here that block–diagrams are geared towards discrete, static systems. Extending such diagrams to discrete, dynamic systems is relatively straightforward, but will involve introducing primitives for temporal delays inside each block. Extending the diagrams to continuous, dynamic systems, however, is non–trivial. One should also note that translating block diagrams into causal networks takes linear time and is technically straightforward. Therefore, for AI researchers, the adoption of block diagrams can be viewed as a syntactic convenience, especially when such blocks are visualized and edited using sophisticated GUIs. But for engineers and end-users with no AI background, the use of block diagrams can have a much more dramatic effect—it can be the decisive factor in whether the model–based technology will be adopted.

3.2 Component libraries

Mapping block diagrams into causal networks relieves system engineers from having to be experienced in the use of causal networks and, therefore, solves the modeling-language problem I discussed earlier. This technique, however, does not address the second problem I identified: the modeling-for-diagnosis problem.

Although engineers feel quite comfortable using block diagrams to model their systems, they will need to know at what level of detail this modeling is to take place. It has been my experience that the required level of detail is a function of the diagnostic coverage of interest, and that identifying the required level can be quite involved and typically requires extensive experience in developing model-based diagnostic applications. I have therefore concluded that, aside from trying to provide engineers with this experience, the only solution to this problem is *to develop component libraries for different domains and provide engineers with these libraries to model their systems.*

Take the avionics domain for example and consider the problem of diagnosing communication failures when transmitting data among different LRUs. For this domain, I have worked with engineers to decide the adequate scope and granularity for system models. The formal outcome of such a process has been the identification of a comprehensive set of components that constitute the building blocks for models developed for this application. For each of these of components, we identified the inputs, outputs, health modes, and behavioral equations, therefore, producing block definitions similar to the one shown in Figure 4. The set of block definitions induce a component library which engineers can use to model different configurations of systems in this application area. Note, however, that the component library is not only system specific, but also class-of-failures specific. That is, if interest arises in diagnosing new system failures, this may lead to an extended (or different) component library. But once such a library is defined, diagnostic models can be developed quite easily for different configurations of the targeted system.

This library-based approach relieves engineers from having to decide on the scope and granularity of their system models, making the modeling process a matter of selecting and connecting system components. But it comes, of course, at the expense of generality. A component library is general enough to handle a system/class-of-failures pair only. Once the system or the failures of interest change, experienced knowledge engineers must get into the loop to extend or amend component libraries as necessary. Note, however, that a change in the failures of interest, for a given system, does not occur too often. Therefore, the compromise worked out between generality and usability has proven to be quite reasonable in practice.

One should point out that the notion of component libraries is hardly new; almost every compositional modeling approach has its own notion of such libraries. Therefore, my message here is not about component libraries *per se,* but about the distinction between those who build such libraries and those who use them to model systems. It is quite reasonable to expect that an average engineer would be capable of modeling a system by instantiating and composing components from a well developed library. But it is not as reasonable to expect such an engineer to build a component library for a particular domain. Building component libraries will typically require reasonable experience with model–based diagnosis as it hinges on subtle choices regarding scope and granularity. Moreover, such choices, as I argued earlier, are specific to the class–of–failures of interest and sometimes demand key insights into the workings of model–based diagnosis.

3.3 Three layers

The solutions to the modeling-language and modeling-for-diagnosis problems led to a three-layer modeling approach which is depicted in Figure 7. Each of the three layers serves a specific purpose and is intended for a certain class of users:

The Causal-Networks Layer is introduced to enable the use of structure-based algorithms whose performance is governed by the connectivity of a system structure. This layer is motivated by performance considerations and is typically accessible to CNETS developers only.

The Block-Diagrams Layer is introduced to shield engineers from having to be informed about causal networks. It constitutes a solution to the modeling-language problem and is typically restricted to knowledge engineers and advanced system engineers.

The Component-Libraries Layer is introduced to shield engineers from having to decide on the scope and granularity of their system models. It represents a solution to the modeling-for-diagnosis problem and is accessible to the average system engineer.

Compositional system modeling is a very desirable paradigm which is sought by many model-based approaches. What distinguishes DTOOL, however, is the ease with which it has encompassed this paradigm. This ease is mainly attributed to the use of causal networks as the underlying modeling language as I have described earlier. I stress, however, that the most important element contributing to the success of DTOOL has not been the notion of compositional modeling *per se,* but the existence of component libraries which has clearly been enabled by such compositionality.

4 Embeddable Diagnostics

DTOOL/CNETS supports two modes of reasoning: interactive and embedded. In the interactive mode, the user poses diagnostic queries to DTOOL which transfers them to CNETS. The results of these queries are

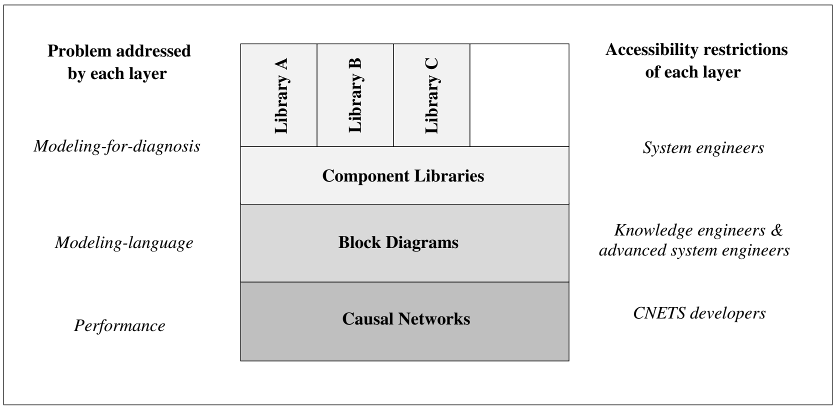

Figure 7: The three layers in the adopted modeling approach. Each layer is intended to address a particular problem and its accessibility is restricted to a certain class of users.

computed by CNETS and handed back to DTOOL which takes on the responsibility of displaying them to the user. Figure 5 depicts an example scenario of this interactive mode and Figure 8 depicts the corresponding information flow. The algorithms underlying this interactive mode are given in [Darwiche 1998b] together with pseudocode for implementing them using a procedural programming language.[4]

This interactive mode is the initial mode of reasoning envisioned for DTOOL/CNETS as it conforms to the mainstream view on how model-based diagnostics should be implemented. This mode is very important as I have found, especially for model development as it allows one to debug the model by posing different queries that explicate the model's behavior.

I have encountered key difficulties, however, when trying to deploy DTOOL/CNETS in a number of applications as it initially supported this mode of interaction only. In these applications, the software and hardware platforms hosting the diagnostic system were so limited that they could not support a complex and demanding system such as DTOOL/CNETS. Even when the platforms were rich enough, they conflicted with the platform for which DTOOL/CNETS was initially developed (Allegro Common Lisp on a PC). One application required implementation on an in-house processor using ADA. Another application required the use of a hardware platform known as a programmable logic controller (PLCs) running a dedicated programming language known as Ladder Logic (LL) [Darwiche and Provan 1996]. The situation was complicated further by the existence of hand-crafted, rule-based diagnostic systems on these platforms, which have been successfully implemented under the mentioned constraints.[5]

Trimming DTOOL/CNETS could have helped in reducing its software and hardware demands, but one could have only gone that far with such a solution. Moreover, developing a new version of DTOOL/CNETS for each platform was clearly cost–ineffective given the complexity of the system and the multiplicity of platforms to support.

To address these problems, I have equipped DTOOL/CNETS with an embeddable mode in addition to its interactive mode. The embeddable mode is based on a technique for compiling a system model into an embeddable diagnostic system that can be easily targeted towards a particular platform [Darwiche 1998a].

[4]The algorithms described in [Darwiche 1998b] are actually more recent than those implemented in CNETS, which are based on [Darwiche 1992]. The algorithms in [Darwiche 1998b] are much simpler and offer stronger complexity guarantees.

[5]From a theoretical viewpoint, it is hard to see how one would prefer a rule-based system over a model-based one. To appreciate this preference, however, consider the following argument which sums up the perspective in industry. Using a hand-crafted, rule-based approach, the problems of correctness, completeness and model-update can all be resolved to a large extent (with high cost and difficulty), but before the diagnostic system is deployed. Once the on-line diagnostic system is deployed, one is guaranteed efficiency and small footprint. And this is exactly what matters as we embed a diagnostic system on an artifact. Contrast this with the model-based approach which simplifies the development of a diagnostic system and provides correctness and completeness guarantees, yet entails an on-line diagnostic system with potential efficiency and footprint problems. The bottom line: Do you pay the penalty in the laboratory — while developing and updating the diagnostic system — or do you pay it on the artifact – as you embed the diagnostic system? As it turns out, in many real-world situations, it makes more sense to pay a one-time cost in the laboratory (a developer's territory) than to pay it on each and every artifact (a user's territory).

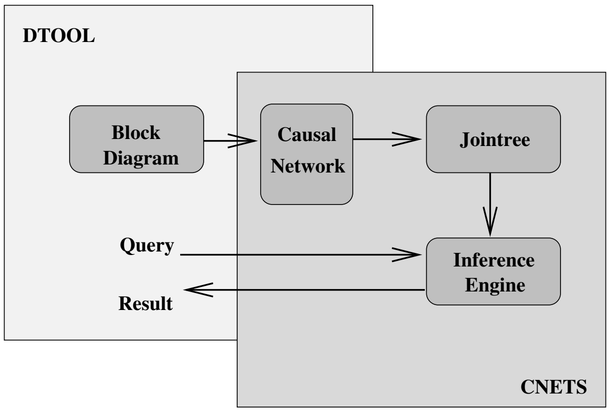

Figure 8: Information flow in the interactive mode of DTOOL/CNETS. Key to the computations underlying this interactive mode is the transformation of a causal network into a jointree, which can be viewed as an aggregation of the system components into a tree structure. The complexity of reasoning in the interactive mode is governed by the properties of the jointree [Darwiche 1998b].

According to this compilation approach, the user

develops a system model in DTOOL;

specifies which component ports will be observed; and

selects a software/hardware platform for the embeddable diagnostic system.

CNETS will then generate an embeddable diagnostic system for the selected platform. Consider Figure 9 for an example. The user has modeled a system which consists of two cascaded inverters by selecting and connecting two instances of the inverter-component from the digital-logic library. The user has also specified that he wishes to collect observations about the input A and output C of this system. He has also selected C++/PC as the software/hardware platform. CNETS will then generate a corresponding diagnostic system which is depicted on the right of Figure 9.

The synthesized embeddable diagnostic system is composed of two components:

A system compilation: This is simply a Boolean expression in decomposable negation normal form [Darwiche 1999b, Darwiche 1998b]. The time to generate the system compilation and the space to store it depend on properties of the system model. Moreover, this Boolean expression must be regenerated anytime the system model or observables change. The compilation, however, is independent of the software/hardware platform.

A diagnostic evaluator: The evaluator is a very simple piece of software that can answer diagnostic queries by operating in linear-time on the system compilation. The evaluator is developed only once for a given software/hardware platform and can operate on the compilation of any system model. Note that CNETS has a number of platform-specific evaluators, one of which is selected and included in the embeddable diagnostic system. The simplicity of the evaluator, however, makes it easy to develop multiple versions of it, each oriented towards a specific platform.

I will now briefly describe the system compilation and the diagnostic evaluator. More detailed descriptions of these components can be found in [Darwiche 1998a, Darwiche 1999a].

Suppose that Δ is a set of Boolean sentences that represent a system model. Suppose further that $\mathbf{O}$ and $\mathbf{A}$ are atoms appearing in Δ, which represent *observables* and *faults,* respectively. If $\mathbf{X}$ are all atoms in Δ, excluding observables and faults, then compiling model Δ amounts to two steps:

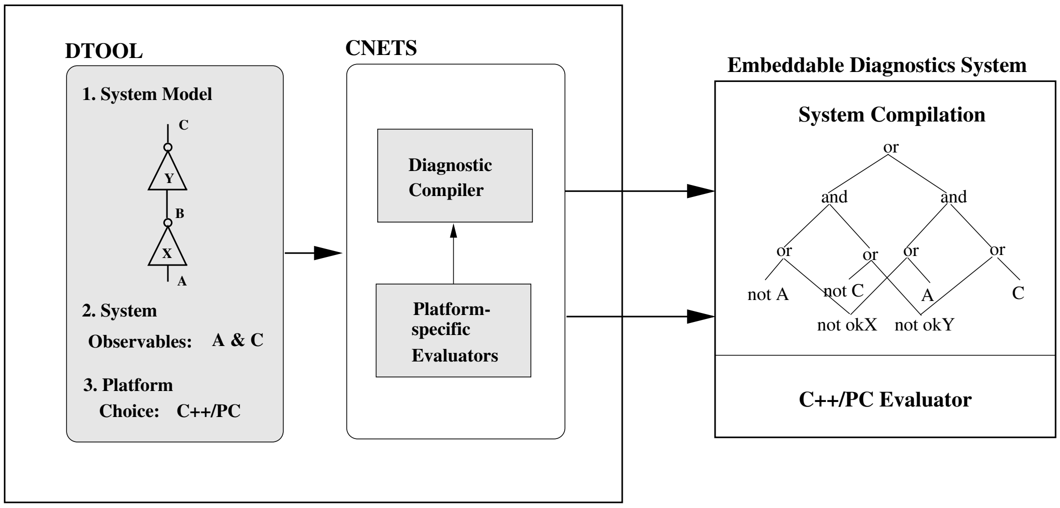

Figure 9: The architecture of the DTOOL/CNETS diagnostic compiler.

Eliminating any reference to atoms **X** from Δ, to yield another model Δ', which is equivalent to Δ as far as answering queries about the observables **O** and faults **A**. This process is known as computing the *consequence* of Δ on **O** and **A** [Darwiche 1998b]. It is also known as *forgetting* about **X** in Δ [Lin and Reiter 1994].

Converting the transformed model Δ' into decomposable negation normal form Δ'' [Darwiche 1999b].

Therefore, the result of compilation is a model Δ'', which (a) mentions only observables and faults; (b) is equivalent to the original model Δ as far as queries concerning observables and faults; and (c) is in decomposable negation normal form (DNNF).

A sentence is in negation normal form (NNF) precisely when it is constructed from literals using the conjoin and disjoin operators. To be decomposable, an NNF must satisfy the following property. Any time a conjunction $\alpha \wedge \beta$ appears in the NNF, the two conjuncts α and β should not share atoms. A DNNF is represented using a rooted DAG, where each internal node represents an operator (and/or) with its children representing operands (conjuncts/disjuncts). Figure 9 depicts a sentence in DNNF.

DNNF is a tractable form. There are more than half a dozen logical operations which can be applied to DNNF in linear time and these operations are sufficient to implement the computational tasks of model-based diagnosis. I describe some of these operations below.

Testing the satisfiability of a DNNF. This operation is all we need to decide whether a certain system behavior is normal (consistent with the given model).

Computing the minimum-cardinality of solutions that satisfy a DNNF. This can be used to compute the smallest number of faults implied by a system observation. Recall that the cardinality of a solution is the number of negative literals in the solution. For example, $\neg A \wedge \neg C \wedge okX \wedge okY$ and $\neg A \wedge \neg C \wedge \neg okX \wedge okY$ are two solutions for the DNNF in Figure 9 with cardinalities 2 and 3, respectively.

Minimizing a DNNF to exclude solutions that do not have a minimum cardinality. This operation can be used to focus on minimum-cardinality diagnoses.

Enumerating the solutions of a DNNF, which can be used to generate diagnoses.

Each one of these operations can be implemented by simply traversing the structure of a DNNF, while performing a simple operation at each node in the DNNF. For example, to compute the minimum–cardinality of any DNNF solution, all we need to do is the following:

For each leaf node in the DNNF, assign the number 0 if the node represents a positive literal, and 1 if it represents a negative literal.

For each and–node in the DNNF, assign the summation of numbers assigned to its children.

For each or–node in the DNNF, assign the minimum of numbers assigned to its children.

The number that gets assigned to the root of the DNNF is then guaranteed to be the minimum–cardinality of any solution of the DNNF. If we apply this process to the DNNF in Figure 9, we get the number 0, which means that the DNNF has a solution with cardinality 0. The solution happens to be $A \wedge C \wedge okX \wedge okY$ in this case.

The diagnostic evaluator depicted in Figure 9 is simply a piece of software that implements the subset of DNNF operations necessitated by the given domain/application. For example, if we want to simply determine the minimal number of faults present in a device given a system observation, then all we need to implement is the procedure for computing the minimum-cardinality of a DNNF. The theoretical underpinnings of the compilation approach and the guarantees it offers on the time to generate (and space to store) a system compilation are detailed in [Darwiche 1999b, Darwiche 1998a, Darwiche 1999a].[6] In this paper, however, I will only stress the two key problems alleviated by this compilation technique:

Multiple-platform support: To generate a diagnostic system for a new software/hardware platform, one needs to perform two steps only. First, implement a version of the diagnostic evaluator which is specific to that platform [Darwiche 1998a, Darwiche 1999a]. Second, add the implemented evaluator to CNETS. The new evaluator will then be coupled with any system compilation to generate an embeddable diagnostic system for that platform. Moreover, the evaluator itself is so simple that it renders this platform-specific implementation fast and cost-effective.

Embeddability of diagnostic systems: Embeddable systems are typically marked by tight, inflexible constraints and interface requirements. The more complex the embedded system is (from an algorithmic and footprint viewpoints) the harder it is to implement it cost-effectively under the previous constraints. The compilation approach I discussed above, however, reduces the on-line diagnostic system to a combination of a Boolean expression and a simple evaluator. This on-line system is marked by its small footprint and its algorithmic simplicity, therefore allowing cost-effective implementations under the previous constraints.

Although the space occupied by a diagnostic evaluator is quite limited and is independent of the compiled system, the storage needed for a system compilation is a function of the system properties. Depending on the system, the size of compilation may or may not be acceptable. To give some feel of the resources required by an embedded system: a compiled LISP evaluator which implements all DNNF operations can take less than 25KB to store. The DNNF itself can range in size from a few tens of KBs to a few MBs, depending on the system size and connectivity, but a few hundred KBs is not uncommon for a system involving a few hundred components. The diagnostic compiler, however, does provide some very specific guarantees on the complexity of this compilation process which are discussed in the following section.

5 The Structure-Based Guarantees

When a sophisticated and complex technology is presented to an engineering audience, one of the first questions asked about it is: How well does it scale? Model-based diagnostics is no exception! Even after beating the modeling and embeddability barriers, the question that popped its head is: Does this really work on real-world systems? Specifically, how long does it take to compile a system and what should we expect

[6] CNETS [Darwiche 1992] is based on a theory of abstract causal networks which supports both symbolic [Darwiche and Pearl 1994] and probabilistic [Pearl 1988] causal networks. Therefore, the compilation technique is also applicable to probabilistic causal networks as is shown in [Darwiche and Provan 1997]. In the probabilistic domain, the compilation is known as a Q-DAG (Query Directed Acyclic Graph). In the symbolic domain, the compilation is known as a CSD (Compiled System Description) following the tradition in model-based diagnosis where a system model is referred to as a *system description* [Darwiche 1998a, Darwiche 1999a]. Since DTOOL supports symbolic causal networks only, compiling a probabilistic causal network requires direct interaction between the user and CNETS.

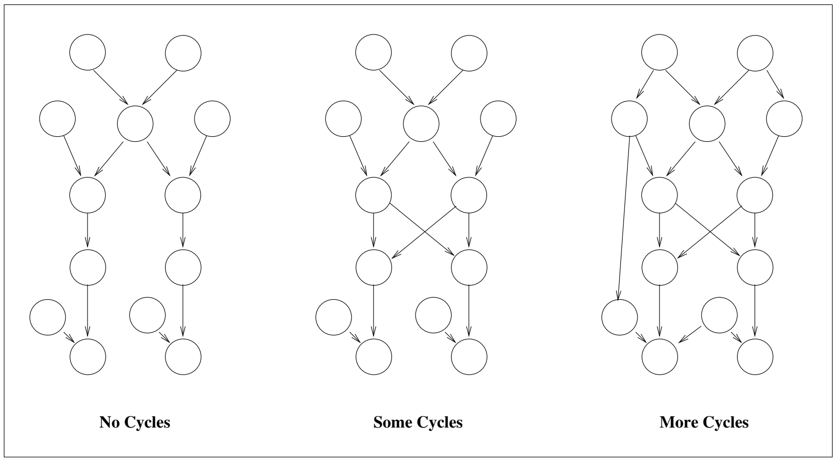

Figure 10: System structures with increasing connectivity. The complexity of CNETS algorithms increase from left to right.

the size of its embedded version to be? Note that answers to such questions will definitely be evaluated in light of what is known about existing systems; rule-based systems in particular.

As is the case for many interesting computational tasks, compiling a system is intractable in general, which means that it is unlikely that we would ever be able to compile all kinds of systems efficiently. This could be quite discouraging, except that one may not be interested in compiling "all kinds of systems!" As it turns out, the compilation technique I have developed is known to be effective on a certain class of systems which is quite meaningful from an engineering viewpoint.

Specifically, the basic algorithms underlying CNETS are structure-based. This means that the amount of work they perform is a function of the system structure on which they operate on. The performance of these algorithms is quite sensitive to the connectivity of the system structure as it will improve as the system structure becomes less connected. For example, if the system structure does not have cycles, the performance of the algorithms is known to scale linearly in the size of the system model. Linear scale-up is quite attractive, but one rarely has systems that are cycle-free. In such a case, the performance will degrade (from being linear) depending on the number of such cycles and how they interact. Figure 10 depicts three system structures with the same number of variables but with increasing connectivity. The performance of CNETS algorithms will degrade as we move from left to right.

The formal connection between the performance of CNETS algorithms and the connectivity of a system structure is given in [Darwiche 1998b] for the interactive reasoning mode and in [Darwiche 1998a, Darwiche 1999a] for the embeddable mode. In a nutshell, the system connectivity is measured using a graph-theoretic parameter known as the *treewidth* [Dechter 1992, Robertson and Seymour 1986, Arnborg 1985]. The CNETS algorithms are known to be exponential only in the treewidth of the system structure and linear in all other aspects of the system. Therefore, if the treewidth is bounded, then compiling a device can be accomplished in linear time and leads to a compilation which can be stored in linear space. Many important classes of structures have a universal bound on their treewidth. For example, trees and forests have a treewidth no greater than 1. Singly-connected directed graphs (ones with no cycles), have a treewidth equal to k, where k is the maximum number of parents per node. Moreover, series-parallel and outerplanner graphs have treewidth no greater than 2 [Arnborg *et al.* 1987]. Structure-based algorithms are now dominant in the probabilistic and constraint literatures where the connectivity of Bayesian/constraint networks drive the performance of various key algorithms in those fields [Dechter 1992, Pearl 1988, Jensen *et al.* 1990].

The version of DTOOL/CNETS discussed in this paper is a prototype developed under Allegro Common Lisp on a PC platform. Its modeling and diagnosis abilities have been demonstrated on a variety of industrial systems, including avionics systems for commercial airplanes, discrete-event systems for manufacturing floors, space-shuttle controllers and in-flight entertainment systems.

My experience with the above industrial applications has confirmed two points. First, that the most critical step in the modeling process is the identification of library components and the level of detail at which they should be modeled. Second, that many realistic systems tend to have manageable treewidths (less than fifteen). I note, however, that it is not uncommon to encounter system structures with large widths (greater than twenty), leading to compilations of unacceptable size. It is my belief that dealing with such systems effectively hinges on extending the current technology in at least one of the following directions:

Utilizing non-structural system properties (system behavior) in the compilation process.
Computational approaches that rely on system structure only are bound not to be optimal as the system behavior plays a crucial role in determining the difficulty of a system. Specifically, two systems may have the same structure, yet one may be easier to, say, compile than the other due to the difference in their behavior. Initial results in this research direction are reported in [Darwiche 1999d].

Trading the completeness of the diagnostic system with its storage requirements.
In rule-based systems, one has full control over the degree of completeness as one is free to include as many rules as one wishes to. Typically, only the most significant rules are included with the unlikely ones excluded. The theory of model-based diagnosis, however, does not provide a parameter for trading the completeness of reasoning with its complexity. This is quite essential, however, as many of the diagnostic queries covered by a complete model-based system concern system behaviors that are too unlikely to materialize, therefore, rendering the completeness property a matter of pure theoretical interest [Davis 1989, Keller 1990]. Initial results in this direction are reported in [Darwiche 1999c].

My current research is focused on addressing the above extensions. One of the key recent developments in this regard has been the introduction of the *device decomposition tree* in [Darwiche 1999d], which is a formal tool for staging structure-based reasoning. I have proposed decomposition trees as an alternative to jointrees, which are the current standard for structure-based reasoning. Using decomposition trees, I was able to exploit system behavior to achieve orders–of–magnitude improvements over purely structure–based approaches. This has been done for the interactive reasoning mode of CNETS and I am currently working on generalizing the technique to the compiled mode.

6 Conclusion

The development of DTOOL/CNETS has been driven by the competition that automated, model-based approaches receive from hand-crafted, rule-based approaches. Although the battle between these approaches has been over in the academic literature for many years now, the situation is different in industry where rule-based systems are dominant and appear to be quite attractive given the considerations of efficiency, embeddability and cost-effectiveness.

I have presented a perspective on this competition and discussed my approach to addressing two of the major barriers facing model-based diagnostics: modeling and embeddability. Specifically, I presented a three-layer modeling approach that renders the process of building system models a matter of selecting and connecting system components. I also presented a compilation approach which allows one to synthesize embeddable diagnostic systems for different software and hardware platforms. Finally, I discussed the formal guarantees that the developed tool offers on the time to generate and space to store embeddable diagnostic systems, showing that both time and space complexity are linear in the device size assuming that the device structure has a bounded treewidth.

Acknowledgement

The DTOOL/CNETS system is the result of many years of work, starting in my Ph.D. thesis at Stanford University, continuing at Rockwell Science Center (RSC), the American University of Beirut, and finally UCLA. Many people have supported and contributed to this effort. My advisor Matthew Ginsberg at Stanford University, Jim Martin at RSC/Palo Alto, and Sujeet Chand at RSC/Thousand Oaks, have all provided much support in this direction. I also wish to thank Gregory Provan of RSC/Thousand Oaks — where the DTOOL/CNETS system was adopted and put to good use — for his continued interest in this system, and our partner Chuck Sitter of Collins Commercial Avionics who strongly believed in this research effort and provided much valuable support. Matthew Barry of Rockwell/Houston Mission Control Center was the first "real-world" user of CNETS. His endorsement of the system in my early days at RSC/Palo Alto was crucial in getting RSC to adopt and nurture model-based technologies. I wish to thank him a great deal for this. Finally, my current understanding of device compilations as reflected in the many references I cite in this paper is owed to the pleasant two years I spent in Beirut, where I had to explain my work to Mathematicians, forcing me to establish an abstract understanding of what started as a down-to-earth exercise in software engineering.

Biography

Dr. Darwiche is an Assistant Professor of Computer Science at the University of California, Los Angeles. He obtained his Ph.D. and M.S. degrees in Computer Science from Stanford University in 1993 and 1989, respectively, and his B.S. degree in Civil Engineering from Kuwait University in 1987. Prior to joining UCLA, Dr. Darwiche managed the Diagnostics and Modeling Department at Rockwell Science Center and was also on the faculty of Computer Science at the American University of Beirut. His research interests include reasoning under uncertainty, model-based reasoning, knowledge compilation, and embedded reasoning systems.

References

[Arnborg *et al.* 1987] Arnborg, S., Corneil, D., and Proskurowski, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Meth.*, 8:277–284.

[Arnborg 1985] Arnborg, S. 1985. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, 25:2–33.

[Darwiche and Goldszmidt 1994] Darwiche, A., and Goldszmidt, M. 1994. On the relation between kappa calculus and probabilistic reasoning. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 145–153.

[Darwiche and Pearl 1994] Darwiche, A. and Pearl, J. 1994. Symbolic causal networks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 238–244.

[Darwiche and Provan 1997] Darwiche, A. and Provan, P. 1997. Query DAGs: A practical paradigm for implementing belief–network inference. *Journal of Artificial Intelligence Research*, 6:147–176.

[Darwiche and Provan 1996] Darwiche, A. and Provan, G. 1996. Exploiting system structure in model–based diagnosis of discrete–event systems. In *Proceedings of the Seventh International Workshop on Principles of Diagnosis*, pages 95–105.

[Darwiche 1999a] Darwiche, A. 1999. Compiling devices: A structure–based approach. Technical Report D–103, Computer Science Department, UCLA, Los Angeles, Ca 90095.

[Darwiche 1999b] Darwiche, A. 1999. Compiling knowledge into decomposable negation normal form. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 284–289.

[Darwiche 1999c] Darwiche, A. 1999. On compiling systems into diagnostic rules. In *Proceedings of the* $10^t h$ *International Workshop on Principles of Diagnosis.*

[Darwiche 1999d] Darwiche, A. 1999. Utilizing device behavior in structure–based diagnosis. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1096–1101.

[Darwiche 1998a] Darwiche, A. 1998. Compiling devices: A structure-based approach. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning*, pages 156–166.

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

[Darwiche 1992] Darwiche, A. 1992. *A Symbolic Generalization of Probability Theory.* PhD thesis, Stanford University.

[Davis 1989] Davis, R. 1989. Form and content in model–based reasoning. In *IJCAI-89 Model–Based Reasoning Workshop*, Detroit, Michigan.

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

[de Kleer and Brown 1984] de Kleer, J. and Brown, J. S. 1984. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83.

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

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

[de Kleer 1986] Johan de Kleer. de Kleer, J. 1986. An assumption-based TMS. *Artificial Intelligence*, 28:127–162.

[Dechter 1992] Dechter, R. 1992. Constraint networks. *Encyclopedia of Artificial Intelligence*, pages 276–285.

[Forbus and de Kleer 1993] Forbus, K. D. and de Kleer, J. 1993. *Building Problem Solvers.* MIT Press.

[Franke and Dvorak 1990] Franke, D. and Dvorak, D. 1990. Cc: Component–connection models for qualitative simulation — a user's guide. Technical Report AI90-126, Department of Computer Science, University of Texas at Austin.

[Franke 1989] Franke, D. 1989. Representing and acquiring teleological descriptions. In *IJCAI-89 Model–Based Reasoning Workshop*, Detroit, Michigan.

[Genesereth 1984] Genesereth, M. R. 1984. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436.

[Hamscher *et al.* 1992] Hamscher, W., Console, L. and de Kleer, J. 1992. *Readings in Model-Based Diagnosis.* Morgan Kaufmann Publishers, Inc., San Mateo, California.

[Jensen *et al.* 1990] Jensen, F. V., Lauritzen, S.L. and Olesen, K.G. 1990. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly*, 4:269–282.

[Keller 1990] Keller, R. M. 1990. In defense of compilation. In *AAAI-90 Workshop on Model–Based Reasoning*, Boston, Massachusetts.

[Kuipers 1994] Kuipers, B. 1994. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge.* The MIT Press.

[Lin and Reiter 1994] Lin, F., and Reiter, R. 1994. Forget it! In *Working notes: AAAI Fall Symposium on Relevance.*

[Pearl 1988] Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., San Mateo, California.

[Reiter and de Kleer 1987] Reiter, R. and de Kleer, J. 1987. Foundations of assumption-based truth maintenance systems: Preliminary report. In *Proceedings of AAAI*, pages 183–188.

[Reiter 1987] Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95.

[Robertson and Seymour 1986] Robertson, N. and Seymour, P. D. 1986. Graph minors II: Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322.