Linköping Studies in Science and Technology

Thesis No. 775

Unique Kernel Diagnosis

by

Anders Henriksson

Submitted to the School of Engineering at Linköping University in partial fulfillment of the requirements for the degree of Licentiate of Technology

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 1999

ISBN 91-7219-514-2
ISSN 0280-7971

Printed by UniTryck, Linköping 1999

Unique Kernel Diagnosis

by

Anders Henriksson

June 1999
ISBN 91-7219-514-2
Linköping Studies in Science and Technology
Thesis No. 775
ISSN 0280-7971
LiU-Tek-Lic-1999:33

ABSTRACT

The idea of using logic in computer programs to perform systematic diagnosis was introduced early in computation history. There are several systems using punch-cards and rulers described as early as the mid 1950's. Within the area of applied artificial intelligence the problem of diagnosis made its definite appearance in the form of expert systems during the 1970's. This research eventually introduced model based diagnosis in the field of artificial intelligence during the mid 1980's. Two main approaches to model based diagnosis evolved: consistency based diagnosis and abductive diagnosis. Later kernel diagnosis complemented these two approaches. Unique kernel diagnosis is my contribution to model based diagnosis within artificial intelligence.

Unique kernel diagnosis addresses the problem of ambiguous diagnoses, situations where several possible diagnoses exist with no possibility to determine which one describes the actual state of the device that is diagnosed. A unique kernel diagnosis can per definition never be ambiguous. A unique kernel diagnosis can be computed using the binary decision diagram (BDD) data structure by methods presented in this thesis. This computational method seems promising in many practical situations even if the BDD data structure is known to be exponential in size with respect to the number of state variabels in the worst case. Model based diagnosis in the form of consistency based-, abductive and kernel-diagnosis is known to be an NP-complete problem. A formal analysis of the computational complexity of the problem of finding a unique kernel diagnosis reveals that it is in P^{NP}.

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Contents

Chapter 1

Introduction

This thesis sums up my work on diagnosis. During my work I have been fortunate enough to be able to follow my own thread of thought without too much interference. The main tool for guidance has been my natural curiosity. If there is something I think I can do better I can not leave it alone.

The combination of these things have led me down many blind alleys during my studies in diagnosis. Very often something may look very easy to improve when you first look at it but once you take a deeper look it becomes clear that it may not be very easy at all. Very often the things I have tried to improve have actually become worse. Those things have naturally been sent to the recycling bin and may very well be part of your yesterdays newspaper.

This thesis is about a few of the things I found that did not end up in the recycling bin.

1.1 Introduction to Diagnosis

To describe ones field of research in a few simple words without getting into details is a challenge. How many times have you been asked by a student or friend about your research and found that your brain goes into an endless loop trying to find an understandable answer. Well, it happened to me a few times.

Anyway here is an attempt to introduce diagnosis, the details will be discussed later.

To introduce diagnosis let us start with the basics. Every now and then something around us goes wrong. This may be more or less anything like windows98 crashing, a person who is not feeling well, a car that will not start in the morning, a space shuttle that blows up, a plane that crashes or maybe even the climate on earth. Diagnosis is to figure out what causes something to go wrong.

The best diagnostician is still in most cases the human expert. Unfortunately the humans who are true experts are rare and have limited capacity while the diagnostic needs are plenty. Thus it would be very nice if a computer program could be used to accurately produce a diagnosis or at least help a human to do so. If this can be done the experts knowledge can be shared with others who can make use of the experts diagnostics skills. This is one of the goals in diagnosis research, put the diagnostic power of the expert in the hands of those who need it.

Another motivation for diagnosis research is that computers can exist and function in places where a human simply can not survive. A computer can be sent to Mars or into places in a nuclear power plant where no human would want to go.

Another big advantage of the computer is that it can go on performing a monotone diagnostic task for years without needing to eat or sleep or rest in any way. This is something a human can never do.

Unfortunately the computer does not only have advantages to offer to humans, a fact that should be well known to most of us. A human expert has the ability to understand and draw conclusions from complex information. This is where the human expert is far superior to any computer. The computer on the other hand is far better when it comes to processing large amounts of simple data. A computer can process this information much faster than a human.

This vital difference between humans and computers makes it virtually impossible for a computer to copy some of the diagnostic tools of a human. This is why finding efficient diagnostic tools suitable to a computer is such a hard problem. This is the

main challenge in diagnosis research.

Diagnosis is closely related to several other fields of research such as verification, debugging, monitoring, fault tolerant systems. The common factor is that in all these cases a vital part of the problem is either to identify a deviance from normal behavior or to find the cause of that behavior. The main difference between these subjects is where the deviance is expected to be and what action should be taken once it is found.

In monitoring the main goal is to detect deviances between the expected behavior and actual behavior. Here the assumption is that our knowledge of what to expect is correct, any deviances from the expected are reported. The deviances reported by a monitoring task may trigger a diagnosis task.

An example would be when you go to the doctor for a checkup. If everything is normal you can go home happy, if on the other hand something is not normal it is likely that the doctor will use his diagnostics knowledge and experience to diagnose what is causing the abnormal behavior.

Another example is what you are doing when you drive your car. If you think something is not as should be with your car you will probably take it to the mechanic. You are subconsciously monitoring your car.

Monitoring and diagnosis also have a central part in creating some types of fault tolerating systems. A fault tolerant system is a system that is able to perform its tasks despite faults in the system. To achieve this the system needs to monitor itself to detect deviances from normal behavior. If a deviance is detected it is likely that a diagnosis have to be made identifying the cause of the deviance in order for the system to know if it can compensate for the deviance and if so how to do it.

One good example of such a fault tolerant system is the interaction between a multi-engine airliner and its pilot. If for some reason one engine has problems which is threatening to cause worse problems like an engine fire this will be detected by the pilot. Either the pilot detects this himself or the airliners warning system attracts his attention. The pilot quickly diagnoses the problem and knows from his training that since the airliner has other engines he can shut down the problem engine

and keep flying. Together the pilot and the plane becomes a fault tolerant system.

Verification is slightly different from monitoring. In verification the goal is to verify that a systems behavior conforms to a specification. Both the system and specification is assumed to be free from faults, the faults that one is looking for is faults which lead to a violation of the systems specification. The main difference from monitoring is that where monitoring is only interested in if the system is performing as expected, verification is interested in if the system will perform as expected in every case.

When doing debugging on the other hand one is looking for construction faults in the specification. The system is expected to conform to the specification but since the specification is wrong the system will not behave as intended.

Diagnosis has a natural place in all the fields of research mentioned above. Often one of these tasks (monitoring, verification, debugging) are necessary to detect the need for a diagnostic task. Once diagnosis pinpoints the cause of the detected problem, repair or fault tolerance actions can be taken.

1.2 Contributions

The main subject for this thesis is the definition and computation of a diagnosis. This is also where my main contributions to diagnosis research are. Here is a short list of contributions to diagnosis research presented in this thesis.

- A new diagnosis criteria.
- A method for computing a diagnosis according to this new diagnosis criteria.
- An analysis of the computational complexity of computing a diagnosis.

The new diagnosis criteria introduced allows more precise diagnosis statements then the traditional methods. On the other

hand it also demands better quality models and access to observations to be useful.

The computation of a diagnosis according to this new criteria can be done just as efficient as for more imprecise diagnosis criteria.

The last contribution is an analysis of the computational complexity of finding a diagnosis.

1.3 What this Thesis is Not About

This thesis is *not* about modeling for diagnosis. Modeling for diagnosis is a separate field of research which I have not studied. Thus this thesis does not contain any modeling examples.

This thesis does not consider the temporal aspects of diagnosis. Although this is a very interesting branch of diagnosis research posing many challenging questions I have decided this to be out of the scope of this thesis. I do however hope to return to this subject later on.

Similarly I hope to study diagnosis methods based on/using continuous values. But so far this has also been decided to be out of the scope of this thesis.

1.4 The Structure of this Thesis

If you have read everything this far you probably have some idea of what this thesis is about. Now it is about time I tell you where in this thesis you will find what.

This thesis is more or less divided into two parts. Chapters 1 and 2 serve as an introduction to discrete diagnosis and this thesis. Chapters 3, 4 and 5 present my contributions to diagnosis research. Since you have read the introduction we will skip that.

- **Chapter 2 Discrete Diagnosis**
  This chapter is an introduction to discrete diagnosis. If you are new to diagnosis research or even discrete diagnosis it may be worth reading this. If on the other hand you know your diagnosis research you should probably skip this chapter.

- **Chapter 3 Diagnosis Theory**
  This chapter introduces a new diagnosis criteria. The relationship between this new diagnosis criteria and other existing diagnosis criteria is explored. This is one of the main contributions to diagnosis presented in this thesis.

- **Chapter 4 Computing a Diagnosis Using Binary Decision Diagrams**
  This chapter presents a method for computing a diagnosis according to the new diagnosis criteria presented in the previous chapter. This method is based on binary decision diagrams and is to my knowledge new to diagnosis.

- **Chapter 5 Evaluation**
  This chapter presents an evaluation of the methods introduced in this thesis.

Now that you have been properly introduced to this thesis you should be ready to read and enjoy it fully.

1.5 Acknowledgements

I am in debt to Krzysztof Kuchcinski for putting up with my endless writing of this thesis and the endless reading that comes with it. Also Christer Bäckström for allowing me the freedom to follow my own thread of thought and develop the material in this thesis as well as answering endless questions on the theory of computational complexity, something which I am also in debt to Peter Jonson for doing. Thomas Drakengren for keeping me from heading into several dead ends and inspiring me to a healthy doze of critical thinking. Simin for reading and helping me maintain a realistic view on life as a PhD student. My family and friends for keeping me sane during the process of producing this thesis.

Chapter 2

Discrete Diagnosis

In this chapter most of the effort will be spent on discrete diagnosis which is the main subject for this thesis.

2.1 Classifying Diagnosis

One common way to classify diagnosis is to look at the information used to produce a diagnosis. Let us use the human expert as a reference. Humans have an astonishing ability to make use of complex information. Especially humans are experts at selecting relevant information from a huge amount of largely irrelevant data. Computers have problems to do both these things.

Computers on the other hand are very good at processing large amounts of simple data, something they can also do very fast. Unfortunately the number of possible solutions to a diagnostics task suffers from a combinatorial explosion. The number of possible solutions for a computer to consider quickly becomes too large for any computer. In order to reduce the amount of data that the computer has to process different forms of abstraction are used.

The classification presented here is based on different levels of abstraction concerning two fundamental factors in diagnosis; time and data. Some diagnosis systems, generally called tempo-

ral diagnosis systems, consider the effects of time while looking for a diagnosis while others do not. Likewise some diagnosis systems use continuous data to produce the diagnosis and are subsequently called continuous diagnosis systems while discrete diagnosis systems only base the diagnosis on discrete information. All combinations of these two factors exist as described by figure 2.1.

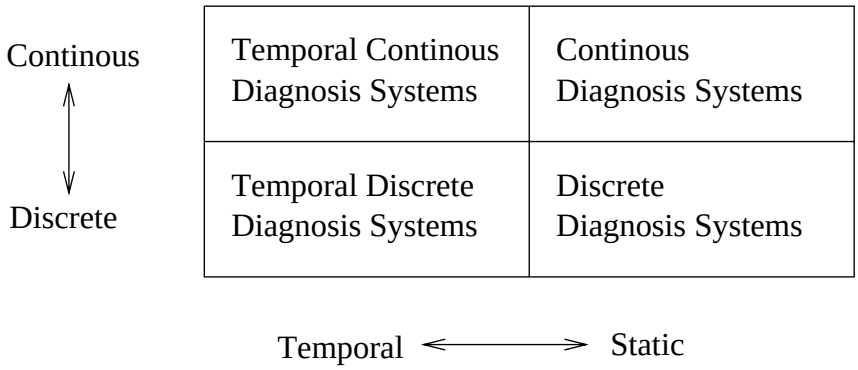

Figure 2.1: Map of Classified Diagnosis Approaches

Temporal continuous diagnosis systems [9, 10, 18, 46, 31] are systems that base the diagnosis on continuous valued information and consider the effects of time on the system to be diagnosed. These diagnosis systems are used to diagnose continuous dynamical systems, a challenging diagnostic problem where values are continuous (not discrete) and values keep changing over time.

These diagnosis problems inspired research that evolved into a separate branch of artificial intelligence research now called qualitative reasoning about physical systems.

One area where these kind of systems are used is in the automobile industry, or actually, in the cars themselves. Here residuals [32, 33] are used in order for the vehicles to comply with the OBD-II[34, 17, 33] emissions regulations. A residual is basically the difference between a measured output from the system to be diagnosed and the predicted output from a mathematical model of the same system.The mathematical model is expected to be correct, thus the residual is a continuous signal

describing how well the system conforms to the expected correct behavior. By studying several residuals sensitive to slightly different changes in the systems behavior faults in the system can be detected and identified [33].

Another type of diagnosis system also used in cars is active diagnosis [3, 44] . Active diagnosis is based on simple tests. By putting the system to be diagnosed in a certain known state and observing the results the diagnosis system can draw conclusions about possible faults in the system to be diagnosed. In order to perform tests in this manner a concept of time is needed in the diagnosis system, together with the fact that most of the observed values from the system (at least in the case of the car) are continuous this method qualifies as a temporal continuous diagnosis method.

Temporal discrete diagnosis systems are systems that base the diagnosis on discrete information such as boolean variables and consider the effects of time. These diagnostic systems are used to diagnose systems where values are discrete or for which a discrete abstraction of significant values is adequate as opposed to continuous values. As with temporal continuous systems values are expected to change over time.

These may be systems where the domain is inherently discrete such as digital circuit [25] or where information from a continuous system is discrete-sized into discrete values [35, 16] .

Diagnosis of systems modeled as discrete-event systems [41, 6, 28]would also be in this category.

Continuous (non-temporal) diagnosis systems are systems that base the diagnosis on continuous valued information but *do not* consider the effects of time. These diagnostic systems are used for diagnosing systems where all values are static in the sense that they do not change over time.

Discrete (non-temporal) diagnosis systems are systems that base the diagnosis on discrete information without regards to time. This category is well explored by several logic-based approaches [37, 39, 12, 11, 29, 38, 13, 20] . These systems are the main subject for this thesis and what moat of this chapter will be about.

2.2 Approaches to Discrete Diagnosis

There are two classical approaches to discrete diagnosis that have dominated the field of discrete diagnosis within artificial intelligence; abductive diagnosis [37] and consistency based diagnosis [39, 22, 12, 13]. At this point it is to early to get into the details, I will get back to this later, the following section is more intended as an overview.

The two approaches are virtually equivalent except for the definition of a diagnosis. An abductive diagnosis is required to support the observations. It must be possible to predict the observed outputs of the system (to be diagnosed) given the inputs to the system together with the diagnosis and a predictive model (system description) of the system. A consistency based diagnosis is considered a diagnosis as long as it does not contradict the observations or the system description.

The differences that do exist are caused by a difference in the use of the system description that originally existed between the two approaches. Abductive diagnosis was originally intended to be used together with fault models. The effects of each fault on the system was to be modeled in the system description, thus the system description should given a correct diagnosis together with the systems inputs be able to predict the observed outputs. This is the basis for the abductive diagnosis criteria.

It has been argued, and there are several relevant points to this argument, that every fault can not be predicted and thus not be modeled. This is however usually not a problem. It is obvious that the abductive approach to diagnosis can only diagnose faults that are known and modeled in the system description. But as humans we never expect to be able to diagnose faults we do not know exists. We simply refer to those faults as unknown, we do not know what is going on only that something is wrong. When an abductive diagnosis system fails to find a known abductive diagnosis it is quite natural to think of this as an unknown fault. Thus it is natural to expect this behavior.

The consistency based approach to diagnosis suffers a similar problem. As mentioned earlier a consistency based diagnosis is considered a diagnosis as long as it does not contradict the ob-

servations or the system description. Unfortunately a diagnosis is not necessarily the correct one in the sense that it describes the actual state of the device only because it is consistent. There may be very many diagnoses that qualify as consistency based diagnoses but only one of them is the correct one.

Both approaches have recognized the advantages of combining model's of the correct behavior with fault models. Several authors have noted the similarities between abductive diagnosis and consistency based diagnosis. This has resulted in several frameworks which include both approaches [8, 29].

The differences were originally based on fault models vs. models of the correct behavior. These were more or less eliminated because it was realized that both approaches benefitted from using both fault models and a model of the correct behavior. Today it seems as if like both approaches were at opposite ends of the same thread. Both approaches are set-covering methods and virtually every computation method end up computing prime implicants.

2.3 The Diagnosis Framework

I believe the best way to present how discrete diagnosis is done is to simply show how. In order to do this we need to build a simple framework. The framework itself is not important, the field of discrete diagnosis is overpopulated by "framework builders" and this is not intended to become another one. Most existing frameworks are very similar and when you have studied one you will easily recognize the others. This framework is presented only for the purpose of an example to explain the structure of the problem.

For illustrative purposes this framework will be applied on a very small example. The simplest of all logic circuits, the inverter.

To start off we will begin with the concept of components and behavior modes[13]. Assume that we have some device $\mathcal{D}$ that we want to diagnose. This device can be anything from an elephant to a space shuttle, what it is is not important here.

What is important however is that we consider this device to be built up from components, thus the device $\mathcal{D}$ can be considered to be a set of components $\mathcal{COMP}$ which it is built from.

In our small example the device consists of a single component which is the inverter, here we name the inverter *inv*.

Now each component is modeled using a set of behavior modes where each mode describes a way in which the component can act. For example the two simplest behavior modes a component could have would be *normal* and *faulty* where normal describes the component to be functioning as it should (however this would be) and faulty describes the component not functioning.

Getting back to our example assuming that the inverter has two possible faults we want to diagnose we can describe the inverter using three behavior modes, one for each fault plus the normal behavior. In this example let us assume that the inverter can fail with it's input stuck at one or stuck at zero, this yields the three modes *normal*, *stuck_at_one* and *stuck_at_zero*.

In a completely analog way we can introduce observations $\mathcal{OBS}$ and observation modes. Observations in $\mathcal{OBS}$ represent the entities we can observe while the observation modes describe the "values" each observed entity can have.

In this example we have two entities we can observe, the inverters input and output. Here we will name these *in* and *out*. Since they are both "digital" signals they both have the same observation modes; *one* and *zero*.

Now we use this to build a set of variables which we will use to build a system description[1] $\mathcal{S}$ of the device. How the variables are constructed or named is of no importance. It is important that we assign one unique variable to represent each combination of component and its behavior modes. We then do the same for observations and get a set of boolean variables $\mathcal{V}$.

For our example we will use a simple naming scheme to achieve this, for every component we construct a variable "component name"_"behavior mode" for each behavior mode. In this example we get three such variables; *inv_normal*, *inv_stuck_at*

[1] A system description is a set of formulas which describe the device

_ *one* and *inv_ stuck_ at_ zero*. Applying a similar naming scheme to our observations we get four observation variables; *in_ one*, *in_ zero*, *out_ one* and *out_ zero*. The complete set $\mathcal{V}$ becomes *inv_ normal*, *inv_ stuck_ at_ one*, *inv_ stuck_ at_ zero*, *in_ one*, *in_ zero*, *out_ one* and *out_ zero*.

These variables can be used to represent the state of the device. A variable is true if and only if the component is working according to the behavior mode which the variable represent. Observations again work in a completely analog fashion. Now the state of the device can be described in a compact form by an assignment $\mathcal{A}$ of values to the variables in $\mathcal{V}$. This assignment can be expressed as an conjunction of literals that when evaluated evaluates to true if and only if the concerned variables have their assigned values. An assignment does not necessarily have to assign a value to every variable in $\mathcal{V}$, if it did it would be quite useless later on in this thesis.

In our example we can describe the inverter being in normal mode with the assignment being a formula consisting of a single literal *inv_ normal*.

We can now describe the system using a set of first order sentences, a system description. For example if the variable o_1 represents some observation having a certain value and the variable c_1 representing a component being in a certain mode and we know that if we observe o_1 the component can not be in this mode we would describe this with the sentence $o_1 \rightarrow \neg c_1$. For any legal assignment this sentence will evaluate to true. This is a general assumption in many frameworks that each sentence in the system description is expected to evaluate to true for any legal assignment.

We can build a simple system description for our example in this way. We begin with describing the basic "rules" in our system description such as the component can only be in one behavior mode at a time, for this we use the following formula:

$$(inv\_normal \wedge \neg inv\_stuck\_at\_one \wedge \neg inv\_stuck\_at\_zero) \vee$$

$$(\neg inv\_normal \wedge inv\_stuck\_at\_one \wedge \neg inv\_stuck\_at\_zero) \vee$$

$$(\neg inv\_normal \wedge \neg inv\_stuck\_at\_one \wedge inv\_stuck\_at\_zero)$$

This also have to be done for each of the observations, this results in two formulas:

$$(in\_one \wedge \neg in\_zero) \vee (\neg in\_one \wedge in\_zero)$$

and:

$$(out\_one \wedge \neg out\_zero) \vee (\neg out\_one \wedge out\_zero)$$

Once the basic rules are established it is time to describe the relation between observations and behavior. Let us begin with the normal behavior which is described by the following formula:

$$((in\_one \wedge out\_zero) \vee (in\_zero \wedge out\_one)) \rightarrow inv\_normal$$

The "stuck at zero" behavior mode:

$$(in\_one \wedge out\_one) \rightarrow inv\_stuck\_at\_zero$$

Finally the "stuck at one" behavior mode:

$$(in\_zero \wedge out\_zero) \rightarrow inv\_stuck\_at\_one$$

With this last formula the example system description is finished. We now move on towards defining a diagnosis.

The first step is to define a legal assignment. A legal assignment is one that complies with the system description and observations.

Definition 2.1 Legal Assignment
An assignment $\mathcal{A}$ is a legal assignment to a system description $\mathcal{S}$ with the observations $\mathcal{O}$ iff:

$$\bigwedge_{\mathcal{S}} \wedge \bigwedge_{\mathcal{O}} \wedge \mathcal{A}$$

is satisfiable.

Now we have enough structure to define a diagnosis. We will begin with the simplest form which is consistency based diagnosis. In consistency based diagnosis an assignment is a diagnosis if and only if it does not contradict the observations, it is consistent with the observations. This is exactly the definition of a legal assignment, thus any legal assignment is a consistency based diagnosis.

A more complicated definition of a diagnosis is abductive diagnosis. An assignment is an abductive diagnosis if and only if it supports the observed outputs. To formulate this in a definition we must first partition our set of observations into two parts; observations regarding the outputs $\mathcal{O}_{out}$ from the device and other observations $\mathcal{O}_{remain}$. With this distinction an abductive diagnosis can be defined as follows:

Definition 2.2 Abductive Diagnosis
A legal assignment $\mathcal{A}$ is an abductive diagnosis to a system description $\mathcal{S}$ with the output observations $\mathcal{O}_{out}$ and remaining observations $\mathcal{O}_{remain}$ iff:

$$\bigwedge_{\mathcal{S}} \wedge \bigwedge_{\mathcal{O}_{remain}} \wedge \mathcal{A} \models \bigwedge_{\mathcal{O}_{out}}$$

Another diagnosis criteria that evolved out of consistency based diagnosis is kernel diagnosis[11]. In order to define kernel diagnosis we first define the term *covering*.

Definition 2.3 Covering
A conjunction A of literals covers a conjunction B of literals iff every literal in A occurs in B.

Remembering that assignments can be expressed as a conjunction of literals (as was done in previous definitions) we now define a *partial diagnosis* as follows:

Definition 2.4 Partial Diagnosis
A partial diagnosis for a system description $\mathcal{S}$ with the observations $\mathcal{O}$ is a legal assignment $\mathcal{A}$ such that for every satisfiable conjunction of literals α covered by $\mathcal{A}$:

$$\bigwedge_{\mathcal{S}} \wedge \bigwedge_{\mathcal{O}} \wedge \alpha$$

is satisfiable.

This definition of partial diagnosis is remarkably similar to the definition of a legal assignment and thus consistency based diagnosis. Actually the partial diagnosis can be seen as a compact representation for all the conjunctions of litterals it covers which are each a consistency based diagnosis.

Now we are ready to define a *kernel diagnosis*:

Definition 2.5 Kernel Diagnosis
A kernel diagnosis is a partial diagnosis with the property that it is the only partial diagnosis which covers it is itself.

If a partial diagnosis is a compact representation for a collection of consistency based diagnosiss a kernel diagnosis is the smallest such representation. One important fact to note at this point is that the definition of a kernel diagnosis allows several kerneld diagnoses to exist for the same diagnostic case. This can be shown with a very simple example which I will leave as a brainteaser for the interested reader.

Thus finding a diagnosis in discrete diagnosis results in searching for assignments which satisfy the given diagnosis criteria. It should by now be clear to anyone that this will be an exponential problem since there is an exponential number of possible assignments to search.

With this introduction I leave you to the rest of my thesis. Hopefully by now you have a grip on what diagnosis is and especially discrete diagnosis. This introduction may not have been as exhaustive as you expected but this is intentional, if you want more information there is good surveys written[24] that will do the job which I recommend. From now on this thesis will be presenting my own work.

Chapter 3

Diagnosis Theory

This chapter describes and motivates the theory behind a new approach to computing a discrete diagnosis. It is intended as an introduction and as such does not discuss the details of how the diagnosis is computed but rather concentrates on explaining an alternative view of discrete diagnosis and how this view is related to more conventional views of discrete diagnosis.

3.1 Motivation

In general diagnosis is viewed as a search problem. The search traverses the search space (more or less randomly) and uses the diagnosis criteria to filter out legal assignments as diagnoses. Any assignment is a potential diagnosis.

The dominance of the search based approaches is very much due to the diagnosis criteria used. Both the abductive and the kernel diagnosis criteria more or less imply a search based approach searching for prime implicants. The consistency based diagnosis criteria is more general but this is at the cost of weaker diagnosis statements.

Using the search approach the size of the search space is a potential problem. Even if a search method has elaborated heuristics for guiding the search they fall back on exhaustive search when their guidance methods fail. Having 300 variables is

not an unreasonable situation. Searching for a legal assignment for 300 variables, each having two possible values, yields a search space of 2^{300} different possible assignments. Compare this with the estimated total number of atoms in the universe 2^{220}. Even if there were $1,000,000 \approx 2^{20}$ legal assignments the probability of finding one of them is less than the probability of finding a single specific atom in the universe. No search based method can deal with this.

Another problem with commonly used approaches to discrete diagnosis is the answers they provide. In many cases a diagnosis may be unusable simply due to its sheer size. It lacks precision.

This has motivated me to search for a diagnosis criteria that allows efficient computation of a diagnosis and accurate diagnosis statements.

3.2 The Diagnosis Problem as Constraint Satisfaction

Taking another look at the general diagnosis problem described in chapter 2 it is quite easy to see how this can be viewed as a constraint satisfaction problem. The search space is the set of possible assignments. We know from chapter 2 that for a system described by a set of variables $\mathcal{V}$ the size of the search space is $2^{|\mathcal{V}|}$.

The system description can be viewed as a set of constraints limiting which assignments are legal assignments. This does not however reduce the size of the search space since it is not possible to tell which assignments are legal without testing them, it only limits the number of legal assignments.

The observations on the other hand can also be viewed as constraints but these do reduce the search space by fixing the value of some variables. If $\mathcal{O}$ is the observations then $|\mathcal{O}|$ denotes the number of variables with a fixed value due to the observations, thus the remaining search space size is $2^{|\mathcal{V}|-|\mathcal{O}|}$. That is, every observation divides the size of the search space in half.

Now consider the obvious fact that in many cases the sys-

tem description together with the observations will constrain the legal assignments in such a way that certain variables must have a certain value in every legal assignment. In these cases the value of those variables is fixed and the search space can be reduced further. The values for the fixed variables form a *core assignment*.

This assignment does not only reduce the size of the remaining search space but it is also the basis for a very strong diagnosis statement.

3.3 A New Diagnosis Criteria

With a new diagnosis criteria at least the issue of precision can be addressed. The computational complexity remains a challenge. The worst case computational complexity is still exponential. There is however an opening for repetitive diagnosis tasks to become more efficient.

Because the new diagnosis criteria is so closely related to the kernel diagnosis [11] this will be named *Unique Kernel Diagnosis*.

3.3.1 Unique Kernel Diagnosis

The intention is to let the core assignment form a diagnosis. The reason that the core assignment is so attractive as a diagnosis is that it can not be contradicted by any legal assignment.

To formalize the core assignment into a unique kernel diagnosis we will need the help of the following definition which defines the "core" properties.

Definition 3.6 Refutable
Given a system description $\mathcal{S}$ and a set of observations $\mathcal{O}$ an assignment $\mathcal{A}$ is refutable iff there is an assignment $\mathcal{A}'$ such that:

1. *$\mathcal{A}$ and $\mathcal{A}'$ assign the same variables.*
2. *at least one variable is assigned a different value in $\mathcal{A}'$ than in $\mathcal{A}$.*

3. *Both $\mathcal{A}$ and $\mathcal{A}'$ are legal assignments for the system $\mathcal{S}$ given the observations $\mathcal{O}$.*

The effect of this definition is that an assignment is refutable if there exists another assignment such that it is both a legal assignment and in conflict with that assignment.

For the definition of the unique kernel diagnosis we are interested in those assignments that are non refutable. We now define the unique kernel diagnosis as the largest non refutable assignment.

Definition 3.7 Unique Kernel Diagnosis
Given a system description $\mathcal{S}$ and a set of observations $\mathcal{O}$ a legal assignment $\mathcal{A}$ is a unique kernel diagnosis iff:

1. *$\mathcal{A}$ is non refutable.*

2. *All assignments $\mathcal{A}'$ which are covered by $\mathcal{A}$ are refutable.*

A unique kernel diagnosis is a very strong diagnosis statement. There is no legal assignment that contradicts the unique kernel diagnosis thus the unique kernel diagnosis can be considered an undisputable fact. No other diagnosis criteria directly pinpoints the facts of the diagnostics case in this manner. This is also the most important difference that distinguishes *unique* kernel diagnosis from the ordinary kernel diagnosis.

The unique kernel reduces the size of the remaining search space as well. Given a unique kernel $\mathcal{A}$ we denote by $|\mathcal{A}|$ the number of variables whose value are fixed by this kernel. This yields a remaining search space is of the size $2^{|\mathcal{V}|-|\mathcal{O}|-|\mathcal{A}|}$.

3.3.2 Undecidable Variables

Another group of variables which deserve to be treated apart from the rest is which values are unconstrained. One effect of constraining the search space with observations and a unique kernel diagnosis is that the number of effective constraints in the system description is reduced while some of the remaining constraints become more precise. This happens as a result of some variables having values are fixed which causes some of

the constraints to have a fixed value. As a result of this some variables may become unconstrained.

To try to determine the value for an unconstrained variable is pointless. Instead we identify unconstrained variables as undecidable with the following definition:

Definition 3.8 Undecidable Variables
Given a system description $\mathcal{S}$, a set of observations $\mathcal{O}$ and a unique kernel $\mathcal{A}$ the set of undecidables $\mathcal{U}$ is defined as the set of those variables such that there is no superset $\mathcal{O}'$ of $\mathcal{O}$ for which the corresponding unique kernel $\mathcal{A}'$ assigns any variable $u \in \mathcal{U}$.

In other words, there is now way to constrain the remaining search space such that the value of the value of the variables in $\mathcal{U}$ can be determined. Once the undecidables have been identified the remaining search space is again reduced since we no longer have to find values for the undecidable variables. The size of the remaining search space is now $2^{|\mathcal{V}|-|\mathcal{O}|-|\mathcal{A}|-|\mathcal{U}|}$

3.3.3 Extensions

Since a unique kernel diagnosis rarely is a complete assignment we define extensions. An extension is an assignment that together with a unique kernel is a complete consistent assignment for every variable that is not undecidable. This is reflected by the following definition.

Definition 3.9 Extension
Given a system description $\mathcal{S}$, a set of observations $\mathcal{O}$, a unique kernel $\mathcal{A}$ and a set of undecidables $\mathcal{U}$ the assignment $\mathcal{E}$ is an extension iff:

1. *$\mathcal{A} \wedge \mathcal{E}$ assigns all variables in $\mathcal{V} - \mathcal{U}$.*

2. *$\mathcal{S} \wedge \mathcal{O} \wedge \mathcal{A} \wedge \mathcal{E}$ is consistent.*

While there is only one unique kernel for each diagnostics case there may be several extensions.

3.3.4 Unique Kernel Diagnosis and Kernel Diagnosis

The unique kernel turned out to be so very similar to the ordinary kernel that it is convenient to consider the unique kernel a variant of the ordinary kernel.

There is actually only one very simple but crucial difference between the ordinary kernel diagnosis and the unique kernel diagnosis; while there can be several ordinary kernel diagnoses there can be only one single unique kernel diagnosis.

The ordinary kernel diagnosis criteria allows multiple kernels to exist as long as they do not cover each other. The unique kernel on the other hand requires the kernel to be the largest unrefutable assignment which makes it unique.

3.3.5 Unique Kernel Diagnosis and Consistency

The relationship between unique kernel diagnosis and consistency based diagnosis is quite trivial. A unique kernel diagnosis is also a consistent diagnosis since it is per definition a legal assignment. This can be summed up with the following theorem.

Theorem 3.1 The Unique Kernel is Consistent
If an assignment $\mathcal{A}$ is a Unique Kernel Diagnosis then $\mathcal{A}$ is also a Consistent Diagnosis.
Proof
Trivial, since $\mathcal{A}$ is per definition a legal assignment.

This knowledge is important because it is necessary for a unique kernel diagnosis to be consistent for it to be able to have extensions. Extensions are also per definition consistent.

3.3.6 Unique Kernel Diagnosis and Abduction

The relation between unique kernel diagnosis and abduction is not as straightforward as that between unique kernel diagnosis and consistency based diagnosis. A unique kernel diagnosis is *not* necessarily an abductive diagnosis and an abductive diagnosis is *not* generally a unique kernel diagnosis. The main reason

is that there is no guarantee that the "outputs" of the system can be predicted given the "inputs" and a unique kernel.

The unique kernel diagnosis and the abductive diagnoses for the same diagnostics case have to be consistent with each other. The reason for this is quite simple, an abductive diagnosis have to be a legal assignment and no legal assignment contradicts the unique kernel diagnosis.

Theorem 3.2 The Unique Kernel is Consistent with Abduction
If an assignment $\mathcal{A}$ is a unique kernel diagnosis and $\mathcal{A}'$ is an abductive diagnosis then $\mathcal{A} \wedge \mathcal{A}'$ is consistent for any abductive diagnosis $\mathcal{A}'$.
Proof
We prove this by refutation. Assume that $\mathcal{A}$ is a unique kernel diagnosis and $\mathcal{A}'$ is an abductive diagnosis inconsistent *with $\mathcal{A}$. Since $\mathcal{A}'$ has to be a legal assignment this means that $\mathcal{A}$ can not be a unique kernel since it is now refutable. Thus if $\mathcal{A}$ is a unique kernel it must be consistent with $\mathcal{A}'$.*

There is no similar relation between extensions and abduction even though every extension is a possible abductive diagnosis. The reason for this is that extensions per definition are consistent with the unique kernel diagnosis.

It is possible to extend the unique kernel definition to a unique *abductive* kernel. This is done in the same way as it is done for the ordinary kernel diagnosis [11] by simply adding the requirement that from the unique kernel and its extensions given the "inputs" of the system it should also be possible to predict the "outputs" of the system.

In order to do this the observations $\mathcal{O}$ must be partitioned into two parts; one part $\mathcal{O}_{out}$ contains observations concerning the "outputs" of the system while the remaining observations not concerning outputs are placed $\mathcal{O}_{remain}$. Now the unique abductive kernel diagnosis can be defined as follows.

Definition 3.10 Unique Abductive Kernel Diagnosis
Given a system description $\mathcal{S}$, a set of observations $\mathcal{O}$, a unique kernel $\mathcal{A}$ and a set of undecidables $\mathcal{U}$ and the extensions $\mathcal{E}$ then

$\mathcal{A}$ is a unique abductive kernel diagnosis iff for every extension $\epsilon \in \mathcal{E}$:

$$\mathcal{S} \cup \mathcal{O}_{remain} \cup \mathcal{A} \cup \epsilon \models \mathcal{O}_{out}$$

The unique abductive kernel diagnosis criteria is even more restrictive than the unique kernel and ordinary abductive kernel. The "outputs" have to be predicted for every extension to the unique kernel or else it is not considered a unique abductive kernel. This is analog to the ordinary abductive kernel definition.

3.4 Consequences of the Unique Kernel Diagnosis

There are some consequences of using a unique kernel diagnosis approach that deserves to be mentioned.

3.4.1 Diagnosis Statements

One of the most precious consequences is the possibility to make definite and precise diagnosis statements where more traditional approaches only can give a list of possible diagnoses. Given a unique kernel it is possible to say that the variables assigned by the unique kernel must have the assigned value, this is an unrefutable truth. If any variables for some reason could have a value other then that assigned by the unique kernel this would indicate a modeling error, there is something about the system that is not described properly in the system description.

In the same way it is possible to make a definite statement that nothing can be said about the value of the undecidable variables. There is simply not enough knowledge represented in the system description to make any such statement.

Likewise it is possible to say the there is not enough knowledge about the current diagnostics case (observations) to make any definite statements about the rest of the variables even if the knowledge is there in the system description.

3.4.2 Extensions

The trick with the unique kernel diagnosis is to separate variables for which definite statements can be made about their value from the rest. The obvious consequence is that for most cases we get a group of variables for which no definite statement about their value can be made. This is where extensions come into the picture.

Extensions are simply the legal assignments for the rest of the variables. Obviously extensions suffer from the same problem as traditional diagnosis approaches, they are only a list of alternative assignments. As such they are equally bulky and expensive to compute.

For this reason it should be considered for each application if and when extensions are relevant and should be computed.

3.4.3 Requirements

The advantages comes at a price. This approach places heavy demands on the quality of the system description and the observations.

To be effective the unique kernel should be as large as possible while the undecidables and the extensions should be few. This depends mostly on how complete the system description is and whether there are enough relevant observations.

Without a good system description it is likely that there will be plenty of undecidables and large extensions while the unique kernel itself is likely to be small. Under these circumstances this approach will be inefficient, but then again so will other approaches.

Chapter 4

Computing a Diagnosis Using Binary Decision Diagrams

Binary decision diagrams (BDD) was invented by Akers [1] in 1978. Ordered binary decision diagrams (OBDD) where introduced by Bryant [4] in a paper from 1986 together with methods for efficient manipulation. Today BDD is used as a synonym for OBDD since the unordered BDD is very little used.

BDD's have proven successful in representing and manipulating boolean functions symbolically in a variety of applications. For this reason I have become interested in exploring the use of BDD's in discrete static diagnosis. This chapter reports my findings in how BDD's can be used for computing a unique kernel diagnosis.

4.1 Binary Decision Diagram Theory

Before discussing using BDD's for diagnosis an explanation of BDD's and their most important properties is in place. To do this I will use a simple example.

4.1.1 What is a BDD

A BDD is a simple data structure that sometimes provides an efficient representation of a binary decision tree (BDT).

A BDT represents a boolean function in a similar way to a function table listing the functions value for each possible assignment of values to the variables in the function. A BDT is a full binary tree with the same number of levels as the number of boolean variables in the function plus one level of terminal nodes (the leafs). Each path from the root to a leaf represents an assignment of values to the variables in the boolean function, the leaf is marked with the functions value for that assignment. A more formal definition of a binary decision tree would be as follows:

Definition 1 Binary Decision Tree
A Binary Decision Tree is a full binary tree where:

1. *Every node is either a decision node (inner node) or a terminal node (leaf).*
2. *Every decision node is marked with a boolean variable name and has two children, one false child and one true child.*
3. *Every terminal node is marked with either true or false.*
4. *Every decision node in the same level is marked with the same boolean variable name.*
5. *Every boolean variable name in the path from the root to any terminal node is unique in that path.*

This representation of a boolean function is significantly more efficient then the function table in terms of storage space. The BDT for a boolean function with n variables can be stored in a storage space proportional to $2 * 2^n$ while the function table requires a storage space proportional to $n * 2^n$.

To visualize this we will use a simple example. Imagine that we want to represent the following formula:

$$((a \wedge \neg b) \vee (\neg a \wedge b)) \wedge (b \vee c) \wedge (c \vee d)$$

a	b	c	d	f(a,b,c,d)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table 4.1: Table representing the example formula

This formula can be represented by a table (table 4.1.1).

If we represent the example formula using a BDT instead it would look like figure 4.1. In this figure the left child is the false child and thus the right child is the true child. To find the evaluated value for a given variable assignment one follows the corresponding path in the BDT and reads the value of the leaf you end up on. For example given the variable assignment $a = 0, b = 1, c = 0, d = 1$ the value of the formula is found by beginning at the root. Since $a = 0$ we follow the left branch from a to b. Since $b = 1$ we follow the right branch from b to c. Following the same pattern we take a left at c and a right at d we end up at a leaf which marks the value to true.

The advantage gained in storage space is at the cost of finding the function value for an assignment. While finding the functions value from a table can be done in constant time it takes time proportional to the number of variables for the BDT.

Even if a BDT is significantly smaller than the corresponding

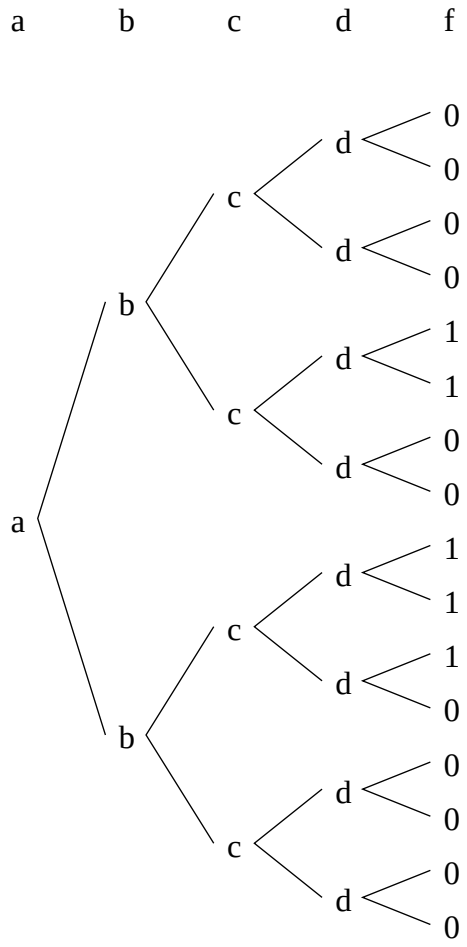

Figure 4.1: The BDT representing the example formula.

table it is still very large. A BDD is a compact representation of a BDT that in many cases is significantly smaller then the corresponding BDT. Bryant [4] showed that a subset of boolean functions can be represented by a BDD that is proportional to the number of variables in the function. It is important to remember however that in the worst cases the size of the BDD is still comparable to that of the BDT.

The BDD data structure makes use of "don't care" in the corresponding function table. A "don't care" is when the value of the boolean function is independent of a certain variable's (or subset of variables) value. For example if we have the following two lines in the table:

$$101 \rightarrow 1$$

$$100 \rightarrow 1$$

Both these lines can be compacted into one with a "don't care":

$$10x \rightarrow 1$$

Since the value of the function is independent of the value of x. These "don't cares" become "shortcuts" in the corresponding BDD graph thus truncating a lot of duplicate nodes in the graph, This is why it is more efficient then BDT's. A BDD can be defined as follows:

Definition 2 Binary Decision Diagram
A Binary Decision Diagram (BDD) is a rooted directed graph with a node set V where:

1. *Every node is either a decision node (inner node) or a terminal node (leaf).*

2. *Every decision node is marked with an index and has two children, one false child and one true child.*

3. *Every terminal node is marked with either true or false.*

4. *For every decision node v index(v) < index(child(v)).*

This definition can be found in Johan Gunnarssons thesis [23] in a more general form. In the following text the false (true) child of s is denoted by child(s, false) (respectively child(s, true)).

Now by a series of definitions we will make the BDD a canonical form. The first step is to define isomorphic BDD's.

Definition 3 Isomorphic Binary Decision Diagram
The BDD's G and G' are isomorphic if there exists a one-to-one function σ from nodes in G to nodes in G' such that if $\sigma(v) = v'$ then:

1. *Both v and v' are terminal nodes with $value(v) = value(v')$.*

2. *Both v and v' are decision nodes with $index(v) = index(v')$, $\sigma(child(v, true))=child(v', true)$ and $\sigma(child(v, false))=child(v', false)$.*

Definition 4 Subgraph
For any node v in a BDD G the subgraph rooted by v is defined as the node itself and all of it's descendants.

Now we can define a reduced BDD as follows:

Definition 5 Reduced Binary Decision Diagram
A binary decision diagram G is reduced if it contains no nodes v where:

1. *$child(v, true) = child(v, false)$.*

2. *distinct nodes v, v' such that the subgraphs rooted by v and v' are isomorphic.*

These definition lead to the following theorem:

Theorem 1 Reduced BDD is a Canonical Form
For any function f over a finite domain there exists a unique reduced binary decision diagram denoting f and any other decision diagram denoting f contains more nodes.
Proof
The theorem is proved by Bryant [4] for the boolean case.

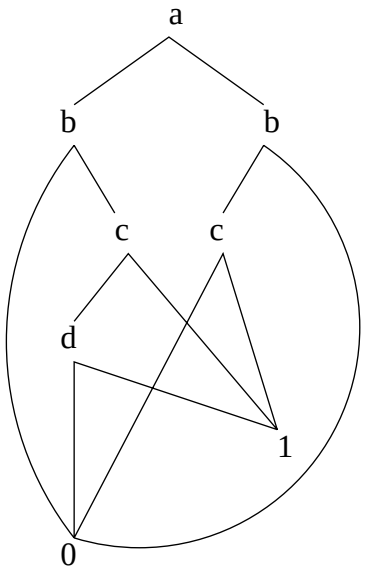

Figure 4.2: The BDD representing the example formula.

If we return to our example formula the corresponding BDD would look like figure 4.2. In a BDD it may no longer be necessary to consider the value of every variable but the value of the formula is still found in the same way as for the BDT.

At this point BDD's seems very promising. A reduced BDD is a canonical representation of a boolean function. As always there is a catch.

The definitions of a BDD and reduced BDD does not impose any restrictions on the variable order. Thus a BDD is only a canonical representation *with respect to a given variable order.* Reduced BDD's with different variable ordering may very well have different size.

The most important part of this catch is that deciding what is in the worst case achieved by performing an exhaustive search.

Another "flaw" is that for some functions the corresponding BDD will be of exponential size for every variable ordering. One such example is the output functions for a combinatorial multiplier [5]. To overcome this problem there has been what Bryant calls "the acronym soup" of proposed variants of BDD's [5]. As far I know none of them succeed in the general case.

4.1.2 BDD Construction

Building the corresponding BDD for a boolean function can generally be separated into two parts: finding a decent variable ordering and building the BDD graph.

As previously mentioned finding the optimal variable ordering is a very exhaustive project. For this reason most approaches rely on heuristic methods for finding a decent variable ordering.

Another approach is Rudell's dynamic variable ordering [40]. This approach is capable of dynamically reorder variables in a BDD and works on computer idle time to experiment with different variable orderings to reduce the size of the BDD.

The approaches to building the BDD graph given a variable ordering can largely be divided into two groups: those traversing the BDT breadth first and those using a depth first strategy. Depth-first was the original strategy and has the benefit of a small memory overhead[1]. Breadth-first approaches are generally faster but suffer from a larger memory overhead.

A third smaller category is the hybrid approach [7]. This approach makes use of the breadth-first speed until a certain memory-overhead threshold is reached when it makes use of the more memory efficient depth-first approach to prune the search tree and lower the memory overhead. Once lowered the breadth first approach can be used again.

Since there is no "superior" or perfect general method for BDD construction there is little motivation for describing any of the methods in detail here. There is an abundance of strategies suited for it's own particular applications but they generally behave poorly in the general case. In any application one will have to select a strategy that is most suited to that particular case. Thus we end the discussion on BDD construction here and move on.

[1]Memory overhead is any memory used that do not store nodes in the graph.

4.2 Diagnosis With Binary Decision Diagrams

Now it is time to take a look at how BDD's can be used for computing a unique kernel diagnosis. This discussion is more or less based on the material presented in chapter 3.

In some cases a BDD can be an efficient representation of a boolean function. For the cases discussed in this thesis the system description is in the form of a boolean function, it is a set of first order sentences that for every legal assignment evaluate to true. If these system descriptions can be represented efficiently with a BDD there is a basis for efficient diagnosis of the system.

4.2.1 Preprocessing

Before we can compute a unique kernel we need to make some preparations. The first step is to represent the system description as a BDD. Because of this first step one limitation to this approach is obvious, some system descriptions can not be efficiently represented with a BDD. The only available solution is unfortunately to avoid using BDD's for those cases.

When the system description BDD has been built we get some trivial verification of the quality of the system description. If at this point the true terminal of the representing BDD is unreachable from the root this means that there is an inconsistency somewhere. Normally the system description with observations and assignments should evaluate to true for all legal assignments, in this case the system can never evaluate to true. This inconsistency is in the system description in the form of a modeling error.

If the true terminal *is* reachable this does unfortunately not mean that the system description does not contain any modeling errors. The reachability of the true terminal can only indicate the existence of modeling errors not verify that they do not exist.

4.2.2 Restricting from Observations

For a system description which can be efficiently represented with a BDD we now move on to the actual computation of the unique kernel diagnosis. The next step is to merge the observations into the representing BDD.

Each observation restricts the value of a variable, this restriction needs to be reflected in the represented BDD. This can be done with a restriction algorithm described in [23]. This restriction is then done for each of the observations. If the restriction algorithm for BDD's is called `Restrict_BDD` a simple algorithm for processing observations would look like this:

Restricting from Observations

```
// Abstract algorithm for processing observations
// Input:  S_BDD a reduced BDD
// Input:  O a set of observations
// Output:  S_BDD a restricted BDD
Restrict_Observations( Observations O,
                       BDD S_BDD      ) {
  for ( o ∈ O )
    Restrict_BDD( S_BDD, o ); }
```

Processing Observations

Similar to the preprocessing step if the true terminal of the BDD is not reachable after the restriction of a variable to a certain value this is an indication of an inconsistency. This inconsistency can be either in the system description in the form of a modeling error or in the observations in the form of a faulty observation.

4.2.3 Finding the Unique Kernel

Assuming that the system description and observations are correct the unique kernel can be found from the restricted system description.

The variables assigned in the unique kernel are characterized by the fact that assigning the variable another value will cause the assignment to be illegal i.e. every branch in the BDD for any other value than the one assigned by the unique kernel will lead straight to the false terminal of the BDD (see figure 4.3).

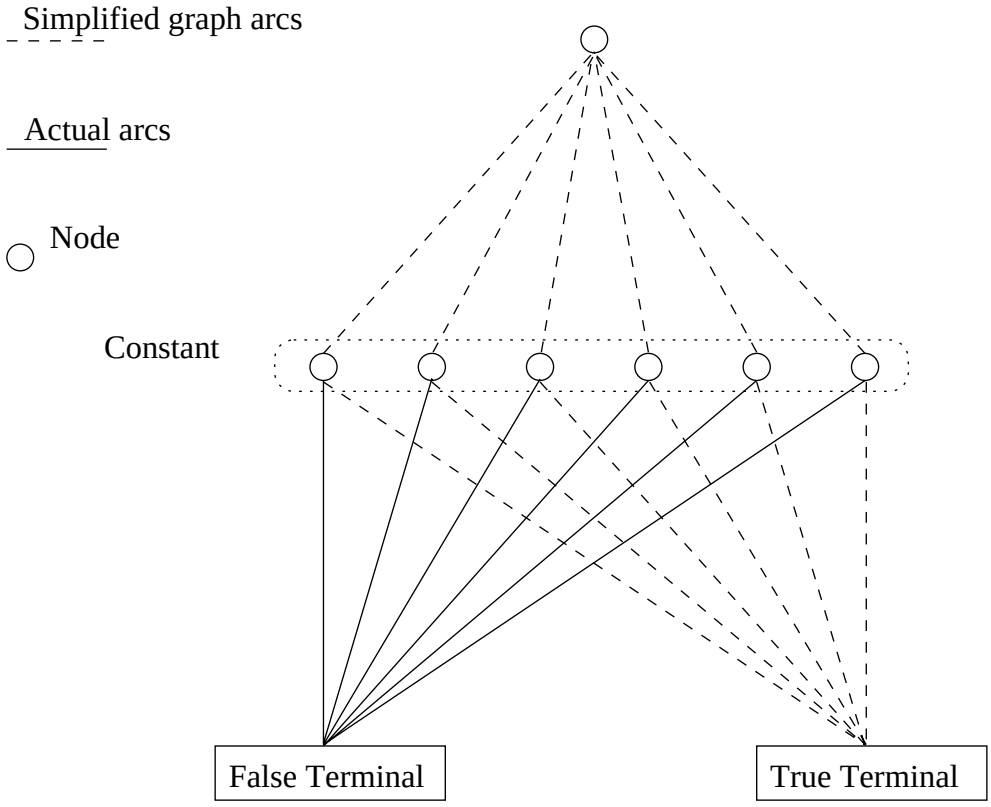

Figure 4.3: Characterization of the Unique Kernel in the restricted BDD

Thus the unique kernel can be found by searching for variables for which all true or all false branches lead straight to the false terminal, those variables must be assigned the opposite value in the unique kernel.

There is however one not so obvious obscurity to watch out for, if there is a branch bypassing the variable that does not lead to the false terminal then the variable can not be part of the unique kernel. The reason for this is simple. Every path from the root of the BDD to the true terminal of the BDD represents a legal assignment. If a node is bypassed this represents that

the value of the node does not matter, the assignment is legal no matter what value is assigned to the variable represented by the bypassed node. Thus any assignment that assigns a value to a variable represented by a bypassed node can be refuted which means that the variable can not be part of the unique kernel. An algorithm for finding a unique kernel would look something like this:

Unique Kernel

```
// Abstract algorithm for finding the
unique kernel
// Input:  S_BDD a restricted BDD
// Input:  V the set of variables in the BDD
// Output:  A the unique kernel
Unique_Kernel ( S_BDD, V, A ) {
  for ( c ∈ V ) {
    if ( c is not bypassed except
       to the false terminal ) {
      if ( ∀false_child(c, S_BDD) ==
                                  false_terminal )
        Append( c, A);
      else if ( ∀true_child(c, S_BDD) ==
                                  false_terminal )
        Append( ¬ c, A); } }
  return A; }
```

Finding the Unique Kernel from a restricted BDD

4.2.4 Finding Undecidables

Finding undecidables is similar to finding the unique kernel, only somewhat simpler. Undecidables are characterized by being unconstrained variables. In BDD terms this means that there are no references to any node representing the variable in the BDD, the variable is simply bypassed.

This can be realized quite easily by considering the consequences of having an unconstrained variable node in a BDD. Per definition since the variable is unconstrained its value can

not affect the value of the boolean expression that the BDD represents. As a result of this the true child and false child of any node representing the variable must be identical. This is however not allowed in a canonical BDD, instead the reference to the disallowed node is set to refer the child of the disallowed node, thus the disallowed node is bypassed (see figure 4.4).

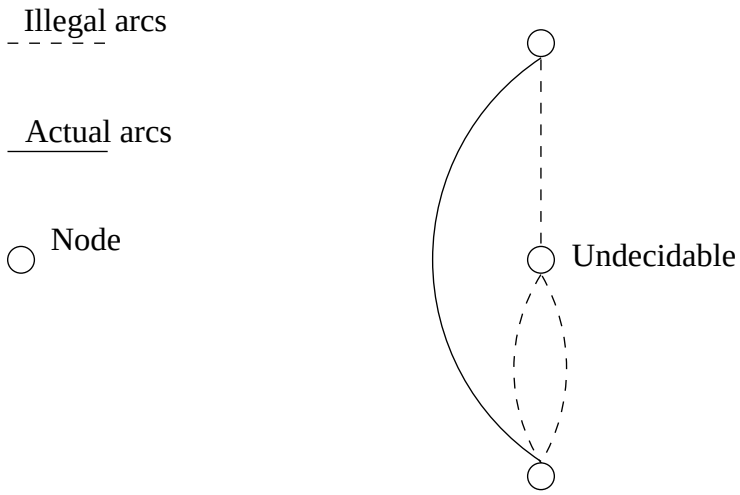

Figure 4.4: How unconstrained variables are bypassed in a BDD

With this knowledge finding the undecidables can be done with a simple filtering algorithm. The algorithm below simply walks the list of variables and filters out those variables which are not referenced in the restricted BDD.

Undecidables

```
// Abstract algorithm for finding the undecidables
// Input:  S_BDD the restricted BDD
// Input:  V the set of variables in the BDD
// Output:  U the set of undecidables
Undecidables ( S_BDD, V, U ) {
  for ( c ∈ V )
    if ( ∀ c ∈ V (c ∉ S_BDD) )
      Append( c, U);
return U; }
```

Finding the Undecidables from a restricted BDD

Finding undecidables can very easily be merged with finding the unique kernel thus performing both in one step.

4.2.5 Finding Extensions

One of the main advantages of a BDD based approach to computing a unique kernel diagnosis is the computation of the extensions. As always one has to decide for each application how important extensions are and whether/how they will be used. This will then determine if the extensions should be computed and if so on what form they should be represented.

The BDD based approach yields a compact representation of the extensions almost as a nice side effect from computing the unique kernel. Only a simple filtering step is needed to produce an extension description.

An extension description is analog to the system description a set of first order sentences which we expect to evaluate to true for each valid assignment. Now consider this extension description to be represented in the form of a BDD. This extension description can easily be obtained from the restricted system description by removing undecidables and unique kernel variables.

Undecidables are as mentioned earlier already bypassed in the restricted system description, thus they are easily removed. The unique kernel variables however have to be filtered out of the restricted system description. This is done by constructing a similar bypass as for the undecidables (see figure 4.5).

This is done with the assumption that the variables in the unique kernel must have the assigned value. Thus every reference to the false terminal is irrelevant since it can not be followed and as such removed. When this is done the unique kernel variables can be bypassed by pointing every reference to a kernel variable to it's remaining child. A simple algorithm doing this would look like this:

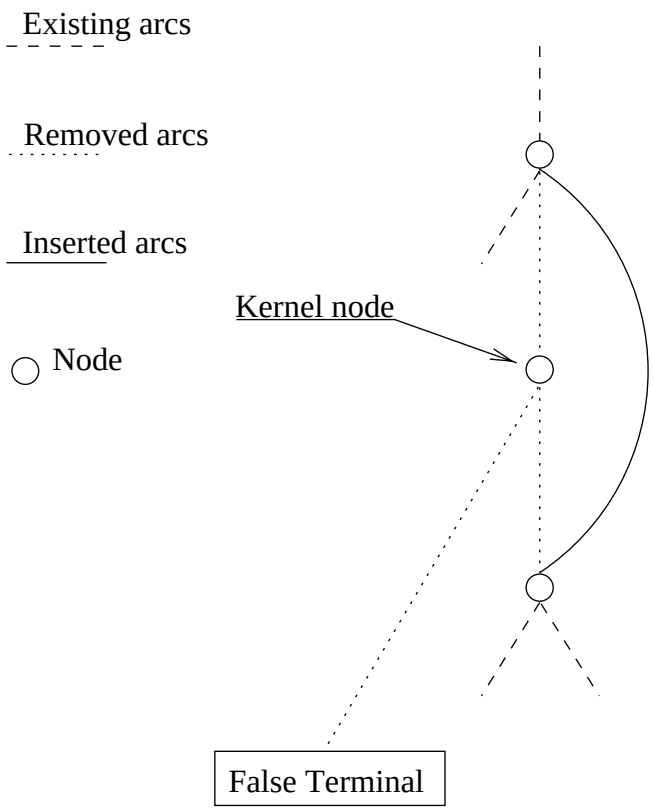

Figure 4.5: Bypassing unique kernel variables in a BDD

Extensions Filtering

```
// Abstract algorithm for bypassing
// kernel variables
// Input:  S_BDD the restricted BDD
// Input:  A is the set of variables in the
// unique kernel
// Output:  E_BDD is the extension BDD
Extension_Filter ( S_BDD, A, E_BDD ) {
  for ( c ∈ A ) {
    for ( n ∈ nodes( c, S_BDD ) {
      remove_ref( n, false_terminal );
      for ( p ∈ parents( n, S_BDD )
        move_ref( p, n, remaining_child(n) ); } }
  return E_BDD; }
```

Bypassing kernel variables.

Unfortunately the result of the above algorithm may not be a proper BDD, it may include nodes for which the true and false children are identical (see figure 4.6).

Fortunately this can be easily fixed by a bottom up algorithm that merges identical subgraphs and bypasses every node with identical children.

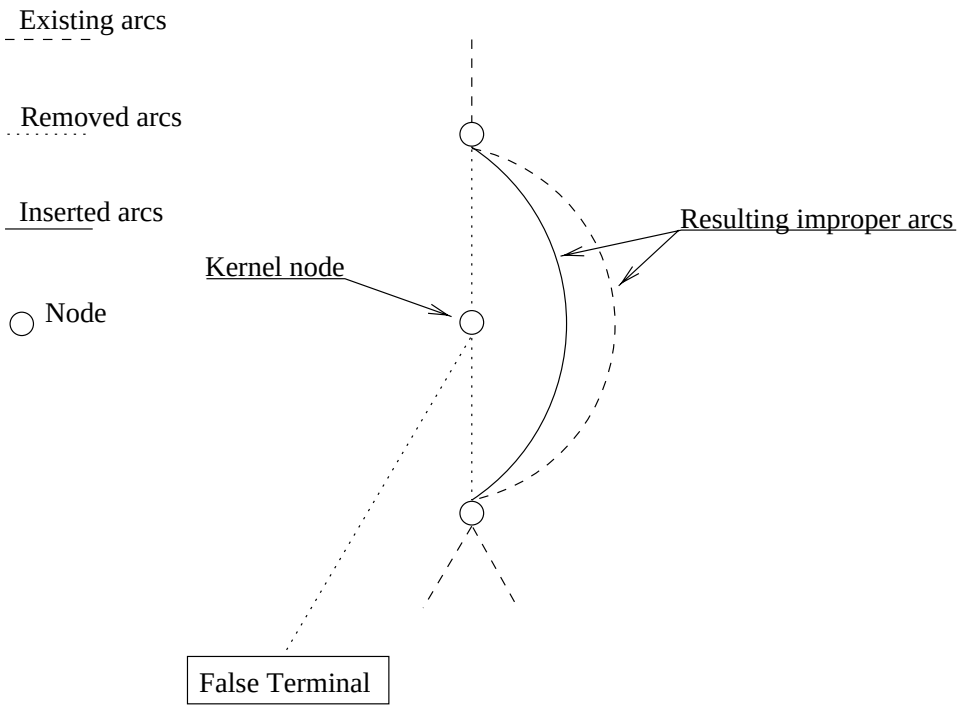

Figure 4.6: Improper nodes in the resulting extension BDD

Extensions Repair

```
// Abstract algorithm for repairing the
// extension BDD
// Input:  E_BDD the extension BDD
// Input:  V is the set of variables in bottom
// up order
// Output:  E_BDD the repaired extension BDD
Extension_Repair ( E_BDD, V ) {
  for ( c ∈ V ) {
    for ( n ∈ nodes( c, S_BDD ) {
      if (true_child( n ) == false_child( n ))
        for ( p ∈ parents( n, E_BDD ) {
          move_ref( p, n, true_child(n) );
          remove( n ); }
      for ( m ∈ nodes( c, S_BDD )
        if (( m != n) &&
            (subgraph( n ) == subgraph( m ))) {
          n = merge( m, n );
          remove( m ); } } }
  return E_BDD; }
```

Repairing the extension BDD.

Now that the BDD is restored we have a compact representation of the extensions. If for some reason it would be necessary to list all the extensions this could be done by a simply traversing the graph. More important it can be the basis for more advanced explorations of the extensions search space.

4.3 Summary

BDD's can be efficient for use in diagnosis in some cases. If BDD's are efficient depends largely on if the system description can be represented efficiently with a BDD.

Finding the unique kernel is generally more costly in a BDD because propagating observations is an $\mathcal{O}(|F|\log(|F|))$ operation. This can generally be done more efficiently. However a canonical representation of the extensions can be constructed with little extra effort where this is generally very costly.

There is an opening for a hybrid approach using constraint propagation to find the core and then building a BDD representation for the remaining system description to represent the extensions. Unfortunately due to the complexity of having to build the extensions BDD "in the diagnosis loop" this approach may be very costly, even so it will beat exhaustive search approaches.

The complexity of finding a diagnosis with this method follows the BDD size. First of all the BDD construction is a preprocessing task and thus does not directly affect the complexity of the diagnosis task. Restricting the representing BDD F with respect to a certain variable assignment is an $\mathcal{O}(|F|\log(|F|))$ operation [4], thus restricting with respect to n observations is an $\mathcal{O}(n|F|\log(|F|))$ operation. Finding the unique kernel by the trivial search method is an $\mathcal{O}(|F|)$ operation. So is eliminating the unique kernel from the BDD. This makes the whole diagnosis task an $\mathcal{O}(n|F|\log(|F|))$ operation. For a reasonable sized BDD this can be very efficient, however in a bad case for a system with m variables the BDD size can be $|F| = 2^m$. If this

is the case $\mathcal{O}(n|F|\log(|F|))$ becomes $\mathcal{O}(n * 2^m)$ which is worse then plain dumb exhaustive search.

This also gives a hint to when the BDD approach can be relied on to be decently efficient. For example if we want to be sure to beat plain dumb search complexity the BDD size must satisfy the following relation:

$$|F|\log(|F|) < (2^m)/m$$

But it is important to remember that these are worst case complexities. Practical cases is a whole different story.

There are a number of BDD related data structures that may be of interest in diagnosis and other applications. One such data structure is BMD's which takes a completely different approach to representing a boolean function. These can sometimes be efficient where BDD's fail[5].

IDD's and IMD's are generalizations of BDD's and BMD's to represent integer valued functions[23]. These have proven to reduce the size of the representing graph in many cases.

Chapter 5

Evaluation

This is an evaluation chapter which is intended to critically evaluate the work presented in the previous chapters. The first section deals with the unique kernel and intends to analyze the computational complexity of the problem of finding a unique kernel. The next section deals with the algorithm for computing a unique kernel presented in chapter 4. The last section contains a general discussion on the material presented in this thesis which is also the last words of this thesis.

5.1 The Unique Kernel

Every now and then we find a hard problem. The problem of finding a unique kernel is such a problem. The natural question is often how hard the problem actually is. To discuss and sometimes answer these questions we have the theory of computational complexity[36, 21]. In this section the problem of finding a unique kernel will be analyzed according to the theory of computational complexity.

5.1.1 The FIX-Problem

In order to analyze the problem of finding a unique kernel we will study a simplified problem which we will call the FIX-problem.

To do this we first need to take a deeper look at the definition of a unique kernel (definition 3.7).

The unique kernel is defined as the largest unrefutable (partial) assignment. An assignment is refutable if and only if there exist another assignment where at least one variable is assigned a different value in the two assignments and both are legal assignments (definition 3.6).

Now let us take a closer look at what is required for a variable to be in the unique kernel. The unique kernel can be created by starting with an empty assignment (an assignment assigning zero variables) and adding variable-assignments which do not make the assignment refutable. When no more variable-assignments can be added the assignment is the largest unrefutable assignment and thus the unique kernel.

The key problem here is to determine if a variable-assignment can be added without making the assignment refutable. This is what we will define as the FIX-problem. To define the FIX-problem we first define what we mean by fix:

Definition 5.11 FIX
Let F be a satisfiable formula involving the variable p. Then p is fix with respect to F iff p has the same value in every model for F.

Then we define the fix problem as follows:

Definition 5.12 FIX-Problem
Let F be a satisfiable formula involving the variable p. Is p fix with respect to F?

The FIX-problem will be used to determine a lower bound for the problem of finding a unique kernel. However in order to build the unique kernel we also need to know what value the variable is fixed to, the fix problem does not yield this information. To determine what value a variable is fixed at we first define what we mean by a variable being fix at a value:

Definition 5.13 FIX(V)
Let F be a satisfiable formula involving the variable p. Then p is fix at a value v with respect to F iff p has the value v in every model for F.

Then we define the fix-at-value problem as follows:

Definition 5.14 FIX(V)-Problem
Let F be a satisfiable formula involving the variable p and v be a value for p. Is p fix at the value v with respect to F?

5.1.2 The Computational Complexity of the FIX-Problem

To establish the computational complexity of the FIX-problem we will study the complementary problem (coFIX-Problem) and prove that this problem is NP-complete thus proving that the FIX-problem is coNP-complete. First we define the coFIX-problem.

Definition 5.15 coFIX-Problem
Let F be a satisfiable formula involving the variable p. Is p not *fix with respect to F?*

It is relatively easy to see that coFIX is in NP. A non-deterministic algorithm only need to find two models for the given formula where the chosen variable have a different value in the two models. If these can not be provided the variable is fixed, if they can be provided the variable is not fixed and this can be verified in polynomial time. Now to prove that the coFIX-problem is NP-complete we show that SAT can be solved by solving coFIX.

Proof: coFIX solves SAT

Let F' be an arbitrary formula.
Let p be a variable that does not occur in F'.
Construct $F = p \rightarrow F'$ (p implicates F').
Case I: F' is sat.
  Assume F' is sat. Then F is sat for any p.
  Thus p can not be fixed with respect to F.
Case II: F' is not sat.
  Case A: Assume $p = 1$.
    $F = p \rightarrow F'$ must be sat (precondition for coFIX).
    $p = 1$ forces F' to be a tautology. But if F' is a tautology
    $p = 0$ is possible which contradicts the assumption.
    Thus this case is impossible.
  Case B: Assume F' not sat.
    If F' is not sat. Then since the implication $F = p \rightarrow F'$ must be sat and F' is never sat this forces $p = 0$ to be fixed.
  Thus p must be fixed with respect to F.
Conclusion:
It is possible to determine if an arbitrary formula F' is sat by constructing a new formula $F = p \rightarrow F'$ where p is a variable that is not in F' and asking coFIX if p is not fix with respect to F. If the answer is yes F' is sat if the answer is no F' is not sat.

The proof that shows how SAT reduces to coFIX and thus how coFIX is NP-Complete

Now we know that the coFIX-problem is NP-complete since we know that SAT is NP-complete. From this information we can also conclude that the complementary FIX-problem is coNP-complete[36].

5.1.3 The Computational Complexity of the FIX(V)-Problem

In the FIX-problem we are only asking if a variable is fix or not, we take no interest in what value the variable is fixed at. The FIX(V)-problem asks a slightly different question asking if a variable is fixed at a specific value. Again we will study this problem by studying it's complementary problem.

Definition 5.16 coFIX(V)-Problem
Let F be a satisfiable formula involving the variable p and v be a value for p. Is p not *fix at the value v with respect to F?*

It is very easy to see that this problem is in NP. The non-deterministic algorithm only needs to provide a model for the function where the variable has the value asked for. The model provided is easily verified in polynomial time. This puts the FIX(V)-problem in NP.

Now it remains to show (co)NP-completeness for the FIX(V)-problem. To achieve this we will "reuse" the proof for NP-completeness of the FIX-problem to show that coFIX(0)[1] can be used to solve SAT. Very little is changed from the original proof.

[1]The case of coFIX(V) when the asked value v=0.

Proof: coFIX(0) solves SAT

Let F' be an arbitrary formula.
Let p be a variable that does not occur in F'.
Construct $F = p \rightarrow F'$ (p implicates F').
Case I: F' is sat.
  Assume F' is sat. Then F is sat for any p.
  Thus p can not be fixed with respect to F.
Case II: F' is not sat.
  Case A: Assume $p = 1$.
    $F = p \rightarrow F'$ must be sat (precondition for coFIX(V)).
    $p = 1$ forces F' to be a tautology. But if F' is a tautology
    $p = 0$ is possible which contradicts the assumption.
    Thus this case is impossible.
  Case B: Assume F' not sat.
    If F' is not sat. Then since the implication $F = p \rightarrow F'$ must be sat and F' is never sat this forces $p = 0$ to be fixed.
  Thus p must be fixed $p = 0$ with respect to F.
Conclusion:
It is possible to determine if an arbitrary formula F' is sat by constructing a new formula $F = p \rightarrow F'$ where p is a variable that is not in F' and asking coFIX(0) if p is not fix $p = 0$ with respect to F. If the answer is yes F' is sat if the answer is no F' is not sat.

The proof that shows how SAT reduces to coFIX(0).

Before we can conclude that coFIX(V) is NP-complete there is one more case to prove, coFIX(1). Fortunately this is completely analog to the proof for coFIX(0) with only a few minor changes. The construction $F = p \rightarrow F'$ (p implicates F') in the proof for coFIX(0) is changed to $F = \neg p \rightarrow F'$ ($\neg p$ implicates F') in coFIX(1) and the variable p's value is swapped in all occurrences in case II. Otherwise the proof remain unchanged.

Now we can conclude that coFIX(V) is NP-complete and thus as before we can also conclude that FIX(V) is coNP-complete.

This information can now be used to establish upper and lower bounds for the problem of finding a unique kernel.

5.1.4 Upper Bound to the Computational Complexity of Finding a Unique Kernel

Now it is time to return to the original problem. Now we have the tools to produce an upper bound for the computational complexity of finding a unique kernel. First we need to define the problem of finding a unique kernel.

Definition 5.17 UK-Problem
Let F be a satisfiable formula and C be a set of variables in F. Find the fixed value for each variable in C that is fixed with respect to F.

It is quite easy to see that UK is unlikely to be in NP. Even if a non-deterministic algorithm could provide a suggested answer that is polynomial in size verifying this answer would require checking it against every model to the formula. Since there may be an exponential number of models this check would be exponential.

It is however possible to solve the UK-problem by a polynomial number of coNP-complete steps. To achieve this we first build a block which determines if a single variable is fixed and tells us at what value the variable is fixed.

UK Building Block

Let F be a satisfiable formula and p be a variable.
Ask two FIX(V) questions for p (FIX(0) and FIX(1)).
Case I: If the answer to both questions is no.
Then the variable p is not fixed at any value and should not be a part of the answer to the UK-problem.
Case II: If the answer to FIX(0) is yes and FIX(1) is no.
Then the variable p is fixed at $p = 0$ and $p = 0$ should be part of the answer to the UK-problem.
Case II: If the answer to FIX(1) is yes and FIX(0) is no.
Then the variable p is fixed at $p = 1$ and $p = 1$ should be part of the answer to the UK-problem.

Building block that decides if a variable is fixed and at what value.

The fourth case in the UK-BB can obviously never occur since a variable never can be fixed to more then one value. The UK-BB is definitely in the polynomial hierarchy [21] since it conforms nicely to the definition of $P^{NP} = P^{coNP}$ which allows a polynomial number of calls (in this case two) to a coNP oracle.

Now all we have to do to produce a unique kernel is to call UK-BB for each variable in the set of variables and save the output. Since there is a polynomial number of variables, in this case the number of variables determines the size of the problem, we have only used a polynomial number of calls to coNP produce the unique kernel. Thus the computational complexity of finding a unique kernel is in P^{NP}.

5.2 The BDD-Based Algorithm

Now let us take a deeper look at the BDD-based algorithm presented in this thesis (see chapter 4) that actually solves the UK-problem. As we have shown in the previous section we should not expect the worst case complexity for this algorithm to be any less then exponential.

5.3 How it works

This algorithm does the classical trade of space for speed and works in a series of steps. The first step is a preprocessing step producing a BDD-representation of the system description. When the BDD is constructed it is restricted with respect to the observations of the current diagnostics case. When the BDD is restricted a search of the BDD finds the answer to the UK-problem.

5.3.1 Constructing the BDD

A BDD is in the worst case exponential in size[4, 5]. This means that there is little hope to find a general algorithm that can

produce any BDD in less then worst case exponential time. In fact it is easy to show that the construction of a BDD is NP-hard since the the SAT problem is trivial to solve by building the BDD for the formula in question. If the only branch from the root of the BDD is to the false terminal the formula is not SAT otherwise the formula is SAT since there exists at least one path from the root to the true terminal.

With a similar reasoning it is equally easy to show how the UNIQUE-SAT problem can be solved by constructing a BDD.

Definition 5.18 UNIQUE-SAT
Let F be a formula. Is it true that there is a unique satisfying truth assignment to F?

This question can be answered by constructing a BDD for the formula and checking if there is a single path from the root to the true terminal of the BDD. Since we know UNIQUE-SAT to DP-complete[36] we know that the construction of a BDD is DP-hard.

With this in mind we can not expect the first step to be anything less then exponential in the worst case. In practice however BDD's have proved themselves to be efficient in many cases, but that is a completely different thing.

5.3.2 Restricting the BDD

The next step is to restrict the BDD with respect to the observations. According to Bryant[4] the worst case complexity of restricting a BDD with respect to a single variable is an $\mathcal{O}(|F|\log(|F|)$ operation where $|F|$ is the size of the BDD in number of vertices. This measure is a little tricky since in terms of the number of variables this would translate into an exponential growth rate since in the worst case $|F| = 2^{|C|}$ where $|C|$ is the number of variables. Restricting several variables individually could possibly be very costly if the number of vertices is high.

Fortunately there is a trivial exponential algorithm that restricts any number of vertices in a single sweep. I have not found

any reference to this algorithm but I am sure it is well known simply due to its simplicity.

The basic idea is to perform several restrictions before reconstructing the BDD properties instead of reconstructing after each restriction. The first step is to perform the restriction.

For each variable that is to be restricted to a certain value for each node representing that variable shift the child reference which represents the *opposite* value to refer to the false terminal. when a variable have been restricted some nodes may get a zero reference count, there are no nodes which reference them. These have to be removed. By doing this in a "top down" fashion starting with the nodes closes to the top some extra work can be saved by avoiding to restrict nodes that will be removed anyway.

When all variables have been restricted the BDD-properties can be reconstructed by the algorithm presented in chapter 4 used for extension repair.

This algorithm is trivially exponential. If one studies the worst case for BDD's which occurs when the BDD "is" a full binary tree until the last level this algorithm will visit each node in the BDD only once. This is generally true for other cases as well. The algorithm works in two phases. The first shifts arcs and removes nodes with zero reference count which are no longer needed, this part is proportional to the number nodes removed. The second phase restores the BDD properties among the remaining nodes of the BDD, as this part also visits every node only once it must be proportional to the number of remaining nodes. Put together this makes the whole algorithm linear in the number of nodes in the BDD, if F is a BDD and $|F|$ is the size of the BDD in the number of nodes the multiple restrict algorithm is $\mathcal{O}(|F|)$. Since the number of nodes in a BDD is in the worst case exponential to the number of variables in the represented expression this makes the multiple restrict algorithm exponential in the number of variables.

5.3.3 Finding the Unique Kernel

Finding the unique kernel is now a search in the BDD. This search is proportional to the number of vertices in the BDD in

the worst case. Even if variables which are bypassed can be skipped in the search thus saving a lot of effort these can not be skipped in the worst case since they can not be expected to exist. Thus the worst case complexity of the search will be $\mathcal{O}(2^{|C|})$.

5.3.4 The Worst Case

Putting everything together gives us a rough estimate of the worst possible situation. Assuming that the BDD can be built by an exponential algorithm, that all the variables have to be restricted (which of course is completely unreasonable) using an algorithm that is at worst $\mathcal{O}(|C|2^{|C|})$ for each variable and that we still have to do the search for the unique kernel (which is also completely unreasonable since all the values for all the variables are known) using an algorithm that is $\mathcal{O}(2^{|C|})$ the whole process is $\mathcal{O}(|C|^2 2^{|C|})$.

Using the multiple restrict algorithm the restrict operation will instead be in $\mathcal{O}(2^{|C|})$ for *all* the restrictions we need to make. This makes the whole process of finding a unique kernel a process of three exponential steps thus it is in $\mathcal{O}(2^{|C|})$.

Both of these estimates are crude over estimates. In any real case after the initial BDD is built the size of this BDD determines the performance of the remaining algorithm. The restriction reduces the size of the BDD, thus it reduces work for the remaining search.

If this solution to the UK-problem would have to be redone every time then this algorithm would be of little use. Practical cases is something completely different.

5.3.5 Practical Cases

For instance if the same system is to be diagnosed several times (which is likely) the BDD would only have to be constructed once. This BDD could then be used to diagnose several cases where the computational complexity of each case would be depending on the size of the BDD.

Restricting the BDD from observations is another very costly operation. This depends on the number of observations and the size of the BDD. If it is known beforehand which observations that will be available it is possible to do this beforehand for a number of frequent cases and then use the pre-restricted version when needed. As usual this is a trades off speed against space.

If the BDD can be constructed and at least partially restricted as preprocessing there is plenty of hope for the remaining restriction of observations and search for the unique kernel to be reasonably efficient.

5.4 Discussion

At last here is a discussion section which concludes this thesis.

5.4.1 The Unique Kernel

The unique kernel does not address the issue of efficiency, at least not when it comes to worst case efficiency. This is apparent from the previous theoretical analysis that finding an efficient algorithm will be very hard since the computational complexity of finding a unique kernel is in P^{NP}. However the problem can be solved by an exponential algorithm based on BDD's presented in this thesis. This more or less makes it equal to other approaches in discrete diagnosis. Since BDD's have proven to be efficient in many other cases of NP-complete problems there is some hope that this algorithm will be efficient in at least some practical cases. This however remains to be proven in further research. Instead the unique kernel focuses on the precision of the resulting diagnosis.

5.4.2 Getting a Useful Answer

In order for a diagnosis system to be useful it must at least provide a useful answer. The user of the diagnosis system must be able to draw at least some useful conclusions from the answer that the system provides. The traditional approaches to discrete diagnosis consistency based diagnosis [39, 22, 12, 13]

and abduction [37] suffer from the same problem; there is in the worst cases an exponential number of possible diagnoses.

This provides us with an intricate problem. If the diagnosis system provides every possible diagnosis the risk is that the answer is useless simply due to its sheer size. Extensive processing of the answer would be needed to draw any useful conclusions if at all possible. If on the other hand the diagnosis system provides us with one of the possible diagnoses as a suggested answer this severely affects the strength of a diagnosis. If the answer is one of possibly an exponential number of answers equally likely to be correct the only possible conclusion is that we know the odds that the provided diagnosis is the answer. Kernel diagnosis [11] suffers a similar problem allowing multiple diagnoses.

The unique kernel diagnosis guarantees a diagnosis that is polynomial in size (linear actually). This means that the answer will never become too large to be useful. Even if some processing of the answer would have to be done there is hope that this could be done efficiently. A unique kernel is also an unrefutable truth with respect to the knowledge we have about the system we diagnosis and the current diagnostic case. The variables given in the unique kernel must have those exact values.

This is also the weakness of the unique kernel. For a system where very little is known about the system and/or there is little knowledge about the facts of the current diagnostics case there will obviously be very few unrefutable facts. Thus the unique kernel will be very small.

There is a distinct tradeoff between different notions of diagnosis with regards to the strength of the answer. This should of course be used when selecting which notion to use. If one is confronted with a system of which very little is known and no good system description is available, then a unique kernel may be very small and due to this fact quite useless. In this case it may very well be better to use a weaker notion of diagnosis like consistency or abduction and process the information this provides. If the system instead is well known and a high quality system description exists then a unique kernel can provide a distinct accurate diagnosis and thus may be a better choice.

To be fair one has to mention that all methods I am aware

of (including the traditional methods mentioned earlier) does become more accurate as the quality of the system description increases. The important thing here is that they do not guarantee a unique diagnosis.

5.4.3 Undecidables and Extensions

It is easy to forget that the unique kernel is not the only diagnosis result that can be found using the approach presented in this thesis. With a little extra effort the undecidables can be found. Where the unique kernel tells us exactly what we know the undecidables tell us what we can never know about the system. This is normally due to the fact that there is not enough knowledge about the system in the system description about the systems current state to draw any useful conclusions. It may in many cases be equally important to know what we can never know as well as to know what we know.

The remainder from the process of finding the unique kernel and the undecidables is the basis for the extensions. Using the BDD-based approach the remaining BDD can be simplified with respect to the unique kernel and the undecidables to produce a BDD describing the extensions. This can now be used to further explore the diagnosis problem.

5.4.4 Combining Set Covering and the Unique Kernel

It would be quite easy to integrate the unique kernel with most common set covering techniques such as consistency based diagnosis, abduction or kernel diagnosis. The unique kernel can be seen simply as a focusing strategy for some other method. For a given diagnostics case one would first establish the unique kernel, the undecidables and the extensions. Then the extensions could be explored using conventional set covering methods thus yielding a combined diagnosis.

One should probably note however that abduction is not a suitable technique in this combination. Abduction relies on the fact that there exists some observation. However all the obser-

vations are "consumed" by the previous steps and not available in the extensions.

5.4.5 Using stochastic methods to find/focus the Unique Kernel

One other interesting approach to finding or at least focusing a unique kernel is the use of stochastic methods. This is founded on the following observation: given two different legal assignments[2] the unique kernel is a "subset" of the variables which are assigned the same value in both assignments. With this in mind it would be possible to search for satisfiable assignments using some stochastic method and maintaining a least common part as a focus.

Even if this seems simple this approach has several problems. The first one is that there is no guarantee in stochastic methods, only probabilities. We may walk around for ever without finding any satisfiable assignment even if it exists. Another problem is when to give up since there is no way to tell when we are finished.

5.4.6 Further Research

The path to efficient unique kernel diagnosis is ... well there is one little problem, we have to show that $P = NP$. Once we do that it will be easy using any approach.

Other than that there are some interesting ideas besides the two mentioned in previous sections. One would be to introduce the unique kernel into an iterative framework where each new unique kernel is used to guide further observations in order to successively bootstrap the unique kernel diagnosis until no further interesting information can be found. Such an iterative process would be interesting to use in combination to residuals to gain access to continuous diagnosis methods, this would strengthen discrete diagnosis in one of it's weakest areas.

Also other methods which can be used to pre-process informations should be looked at. The task of discrete diagnosis seems hopelessly NP. I believe that the only way to make at

[2] These are for example two different consistency based diagnoses.

least repetitive diagnosis tasks efficient is to make the classic tradeoff between space and speed, storing relevant preprocessed information.

5.4.7 Did I succeed?

Did I achieve my goals? Yes and no! I originally wanted to do diagnosis efficiently, this was of course bound to fail in the worst case scenario. I also did not like the structure of the other approaches, they lacked precision. I believe I fixed that. I also wanted this thesis to be less then 50 pages thus avoiding unnecessary junk, we all know by now that I failed that.

Bibliography

[1] S. B. Akers. Functional testing with binary decision diagrams. In *Eight Anual Conference on Fault-Tolerant Computing*, pages 75–82, 1978.

[2] BOSCH. *Automotive Electric/Electronic Systems*. Robert Bosch GmbH, second edition, 1995.

[3] BOSCH. *Automotive Handbook*. Robert Bosch GmbH, fourth edition, 1996.

[4] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[5] Randal E. Bryant. Binary decision diagrams and beyond: Enabling technologies for formal verification. In *International Conference on Computer-Aided Design*, pages 236–243. IEEE, 1995.

[6] Yi-Liang Chen and Gregory Provan. Modelling and diagnosis of timed discrete event systems - a factory automation example. In *Proceedings of the American Control Conference*, 1997.

[7] Yirng-An Chen, Bwolen Yang, and Randal E. Bryant. Breadth-first with depth-first bdd construction: A hybrid approach. Technical report, Shool of Computer Science, Carnegie Mellon University, March 1997.

[8] Luca Console and Pietro Torasso. A spectrum of logical definitions of model-based diagnosis. In Hamscher et al. [24].

[9] P. Dague, P. Devrès, P. Luciani, and P. Tailibert. Analog systems diagnosis. In *Proc. 9th European Conference on Artificial Intelligence*, pages 173–178, 1990.

[10] P. Dague, O. Jehl, P. Devrès, P. Luciani, and P. Tailibert. When oscillators stop oscillating. In *Proc. 12th Int. Joint Conf. on Artificial Intelligence*, 1991.

[11] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56, 1992.

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

[13] J. de Kleer and B.C. Williams. Diagnosis with behavioral modes. In IJCAI11 [26].

[14] Johan de Kleer. Using crude probability estimates to guide diagnosis. *Artificial Intelligence*, 45:381–392, 1990.

[15] Johan de Kleer. Focusing on probable diagnosis. In Hamscher et al. [24].

[16] Dennis DeCoste. Dynamic across-time measurement interpretation. In *Proceedings of the 8th National Conference on Artificial Intelligence*, 1990.

[17] Emissions standards for passenger cars worldwide. Delphi Thecnical Centre Luxembourg, Avenue de Luxembourg, L-4940 Bascharge, Grand Duchy of Luxembourgh, October 1996.

[18] Daniel Dvorak and Benjamin Kupiers. Model-based monitoring of dynamical systems. In IJCAI11 [26].

[19] Michel P. Féret and Janice I Glasgow. A formal model for experinece-aided diagnosis. *Computational Intelligence*, 13(2), 1997.

[20] Gerhard Friedrich and Wolfgang Nejdl. Momo model-based diagnosis for everybody. In Hamscher et al. [24].

[21] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Bell Telephone Laboratories Inc., 191979.

[22] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, November 1989.

[23] Johan Gunnarsson. *Symbolic Methods and Tools for Discrete Event Dynamic Systems.* PhD thesis, Depertment of Electrical Engineering, Linköping University, 1997.

[24] Walter Hamscher, Luca Console, and Johan de Kleer, editors. *Readings in Model-Based Diagnosis.* Morgan Kaufmann Publishers, 1992.

[25] Walter Hamscher and Randall Davis. Diagnosing circuits with state: An inherently underconstrained problem. In *Proceedings of the 4th National Conference on Artificial Intelligence*, 1984.

[26] *Proc. 11th Int. Joint Conf. on Artificial Intelligence*, 1989.

[27] Ronald Jurgen. *Automotive Electronics Handbook.* McGraw-Hill, 1996.

[28] Magnus Larsson. *On Modeling and Diagnosis of Discrete Event Dynamic Systems.* Licenciate thesis, Depertment of Electrical Engineering, Linköping University, 1997.

[29] Peter J.F. Lucas. Analysis of notions of diagnosis. *Artificial Intelligence*, 105:295–343, 1998.

[30] S. McIlraith and R. Reiter. On tests for hypothetical reasoning. Technical Report 92-2, Univ. of Toronto, Dept. of Computer Science, March 1992.

[31] Hwee Tou Ng. Model-based multiple fault diagnosis of time-varying, continous physical devices. In *Proceedings of the 6th Conference on AI Applications*, 1990.

[32] M. Nyberg and L. Nielsen. Model based diagnosis for the air intake system of the SI-engine. Technical Report 970209, SAE, 1997.

[33] Mattias Nyberg. *Model Based Diagnosis with Application to Automotive Engines.* Licenciate thesis, Linköping Universitet, 1997.

[34] California's OBD-II regulation (section 1968.1, title 13, california code of regulations), resolution 93-40, july 9, pages 220.7 - 220.12(h), 1993.

[35] O. O. Oyeleye, F. E. Finch, and M. A. Kramer. Qualitative modelling and fault diagnosis of dynamic processes by midas. *Chemical Engineering Communications*, 96:205–228, 1990.

[36] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company Inc., 1994.

[37] D. Poole. Normality and faults in logic-based diagnosis. In IJCAI11 [26].

[38] Olivier Raiman. The alibi principle. In Hamscher et al. [24].

[39] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32, 1987.

[40] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *International Conference on Computer-Aided Design*, pages 42–47. IEEE, 1993.

[41] Meera Sampath, Stéphane Lafortune, and Demosthenis Teneketzis. A language-based approach to failure diagnosis of discrete event systems. In *Proceedings of the workshop on discrete event systems*, 1996.

[42] Richard Stone. *Introduction to Internal Combustion Engines*. Macmillan Press, second edition, 1992.

[43] Peter Struss and Oskar Dressler. Physical negation - integrating fault models into the general diagnostics engine. In Hamscher et al. [24].

[44] A. Unger and K. Smith. The OBDII system in the volvo 850 turbo. Technical Report 932665, SAE, 1993.

[45] T. Washio, M. Sakuma, and M. Kitamura. A new approach to quantiatative and credible diagnosis for multiple faults of components and sensors. *Artificial Intelligence*, 91:103–130, 1997.

[46] Edward K. Yu. Using constraint logic programing for model-based diagnosis: The modiac system. In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing*, volume I, 1992.

Avdelning, Institution
Division, department

Department of Computer and Information Science

Institutionen för datavetenskap

Datum
Date

June 9, 1999

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English

☐ _____

Rapporttyp
Report: category

☒ Licentiatavhandling
☐ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport

☐ _____

ISBN 91-7219-514-2

ISRN LiU-Tek-Lic-1999:33

Serietitel och serienummer
Title of series, numbering

ISSN 0280-7971

Linköping Studies in Science and Technology

Thesis No. 775

URL för elektronisk version

Titel
Title

Unique Kernel Diagnosis

Författare
Author

Anders Henriksson

Sammandrag
Abstract

The idea of using logic in computer programs to perform systematic diagnosis was introduced early in computation history. There are several systems using punch-cards and rulers described as early as the mid 1950's. Within the area of applied artificial intelligence the problem of diagnosis made its definite appearance in the form of expert systems during the 1970's. This research eventually introduced model based diagnosis in the field of artificial intelligence during the mid 1980's. Two main approaches to model based diagnosis evolved: consistency based diagnosis and abductive diagnosis. Later kernel diagnosis complemented these two approaches. Unique kernel diagnosis is my contribution to model based diagnosis within artificial intelligence.

Unique kernel diagnosis addresses the problem of ambiguous diagnoses, situations where several possible diagnoses exist with no possibility to determine which one describes the actual state of the device that is diagnosed. A unique kernel diagnosis can per definition never be ambiguous. A unique kernel diagnosis can be computed using the binary decision diagram (BDD) data structure by methods presented in this thesis. This computational method seems promising in many practical situations even if the BDD data structure is known to be exponential in size with respect to the number of state variabels in the worst case. Model based diagnosis in the form of consistency based-, abductive and kernel-diagnosis is known to be an NP-complete problem. A formal analysis of the computational complexity of the problem of finding a unique kernel diagnosis reveals that it is in P^{NP}.

Nyckelord
Keywords

Unique Kernel Diagnosis, Model-based Diagnosis

Department of Computer and Information Science
Linköpings universitet

Linköping Studies in Science and Technology
Faculty of Arts and Sciences - Theses

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.

No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.

No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.

No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.

No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.

No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.

No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.

No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.

No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.

No 104 **Shamsul I. Chowdhury: S**tatistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.

No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.

No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.

No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.

No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.

No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.

No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.

No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.

No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.

No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.

No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.

No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.

No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.

No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.

No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.

No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.

No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.

No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.

No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.

No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.

No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.

No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.

No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.

No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.

No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.

No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.

No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge-Bases, 1991.

No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.

No 318 **Lena Strömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.

No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.

No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.

No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.

No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.

No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Sytems, 1992.

No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.

No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.

No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.

No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.

No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.

No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.

No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.

No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.

No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.

No 402 **Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406 **Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414 **Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agentteoretiskt perspektiv, 1994.
No 417 **Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436 **Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
No 437 **Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440 **Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94 **Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94 **Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441 **Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446 **Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450 **Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451 **Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452 **Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455 **Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94 **Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462 **Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464 **Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469 **Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473 **Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475 **Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476 **Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. Fallstudier i tre byggföretag, 1995.
No 478 **Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95 **Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylkprogram, 1995.
No 482 **Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488 **Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489 **Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497 **Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498 **Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503 **Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95 **Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95 **Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513 **Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517 **Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518 **Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522 **Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538 **Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545 **Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546 **Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96 **Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549 **Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550 **Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557 **Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558 **Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561 **Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563 **Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
No 567 **Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575 **Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576 **Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587 **Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589 **Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591 **Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595 **Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597 **Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.

No 598 **Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6 **Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615 **Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629 **Gunilla Ivefors:** Krigsspel coh Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639 **Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640 **Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643 **Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653 **Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674 **Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676 **Jan Håkegård**: Hiera rchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668 **Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675 **Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14 **Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695 **Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
FiF-a 16 **Marie-Therese Christiansson:** Inter-organistorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.
No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader,1999.
No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.
No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.
No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.