DRUM–II: Efficient Model–based Diagnosis of Technical Systems

Vom Fachbereich Elektrotechnik und Informationstechnik
der Universität Hannover

zur Erlangung des akademischen Grades
Doktor–Ingenieur genehmigte

Dissertation

von

Dipl.-Inform. Peter Fröhlich
geboren am 12. Februar 1970 in Würselen

1998

1. Referent:	Prof. Dr. techn. Wolfgang Nejdl
2. Referent:	Prof. Dr.-Ing. Claus-E. Liedtke
Tag der Promotion:	23. April 1998

Abstract

Diagnosis is one of the central application areas of artificial intelligence. The computation of diagnoses for complex technical systems which consist of several thousand components and exist in many different configurations is a grand challenge. For such systems it is usually impossible to directly deduce diagnoses from observed symptoms using empirical knowledge. Instead, the model–based approach to diagnosis uses a model of the system to simulate the system behaviour given a set of faulty components. The diagnoses are obtained by comparing the simulation results with the observed behaviour of the system.

Since the second half of the 1980's several model–based diagnosis systems have been developed. However, the flexibility of current systems is limited, because they are based on restricted diagnosis definitions and they lack support for reasoning tasks related to diagnosis like temporal prediction. Furthermore, current diagnosis engines are often not sufficiently efficient for the diagnosis of complex systems, especially because of their exponential memory requirements.

In this thesis we describe the new model–based diagnosis system DRUM–II. This system achieves increased flexibility by embedding diagnosis in a general logical framework. It computes diagnoses efficiently by exploiting the structure of the system model, so that large systems with complex internal structure can be solved.

We start by developing a novel formalization of model–based diagnosis based on circumscription, a widely used non–monotonic logic. The use of a general circumscription approach makes DRUM–II more flexible than previous diagnosis engines. DRUM–II handles a broad spectrum of diagnosis definitions as well as other forms of non–monotonic reasoning like temporal reasoning. The implementation of circumscription in DRUM–II is based on a new sound and complete search algorithm.

Since model–based diagnosis requires a large number of simulations of the system under similar assumptions, diagnosis engines need techniques to avoid unnecessary recomputations. Previous diagnosis engines which are often based on truth maintainance techniques dynamically record information during the search for diagnoses. In contrast to previous systems DRUM–II uses only static precompiled information about the structure of the system to focus the computation of the diagnoses. The space required for this precompiled information is quadratic in the size of the system under consideration, while the dynamic data structures of previous systems require exponential space. We have measured the running times of DRUM–II on the widely used

ISCAS-85 benchmark circuits. It turns out that DRUM–II shows better performance than previous systems on all circuits for nearly all test vectors.

We demonstrate the use of DRUM–II for alarm correlation in cellular networks. This application is a challenging modeling problem, which can only be solved completely by applying the powerful spectrum diagnosis concept. We show how the computation of spectrum diagnoses is reduced to circumscription by a new iterative algorithm in DRUM–II.

Finally, we consider the process aspect of diagnosis. The computation of diagnoses is a dynamical process during which models on different abstraction levels as well as different simplyfying assumptions are used and measurements are carried out. We define the first declarative modeling language which allows to describe application–specific diagnostic processes. Furthermore we provide an algorithm which executes the process specifications and show it application to a process for hierarchical circuit diagnosis.

Zusammenfassung

Diagnose ist eine der Hauptanwendungen der künstlichen Intelligenz. Die automatische Berechnung von Diagnosen für komplexe technische Systeme, die aus einer großen Zahl von Komponenten bestehen und in vielen unterschiedlichen Konfiguration existieren, stellt eine große Herausforderung dar. In der Regel ist es bei diesen Systemen nicht möglich, durch Anwendung empirischen Wissens direkt von den beobachteten Fehlersymptomen auf die Fehlerursachen zu schließen. Stattdessen wird in der Modellbasierten Diagnose ein Simulationsmodell des zu untersuchenden Systems verwendet, das es erlaubt, das Systemverhalten unter Annahme einer Menge von fehlerhaften Komponenten zu simulieren. Die Diagnosen werden dann durch Vergleich von Simulationsergebnis und beobachtetem Fehlverhalten ermittelt.

Seit der zweiten Hälfte der 80er Jahre wurden mehrere modellbasierte Diagnosesysteme entwickelt. Bisherige Systeme weisen allerdings meist eine eingeschränkte Flexibilität auf, da sie einen eingeschränkten Diagnosebegriff implementieren und der Diagnose verwandte Aufgaben, wie z.B. zeitliche Vorhersagen, nicht unterstützen. Außerdem ist die Effizienz bisheriger Systeme für die Diagnose komplexer technischer Systeme oft nicht ausreichend, insbesondere aufgrund ihres exponentiellen Speicherplatzbedarfs.

In dieser Dissertation wird das neue modellbasierte Diagnosesystem DRUM–II vorgestellt. Dieses System weist aufgrund der Einbettung des Diagnosebegriffs in einen allgemeinen logischen Rahmen eine hohe Flexibilität auf. Darüberhinaus ermöglicht es durch Ausnutzung der Struktur des untersuchten Systems die effiziente Berechnung von Diagnosen und somit die Behandlung großer, strukturell komplexer Systeme.

Zunächst entwickeln wir eine Formalisierung der Modellbasierten Diagnose unter Verwendung der Zirkumskription, einer weit verbreiteten nicht–monotonen Logik. Durch die Verwendung eines allgemeinen Zirkumskriptionsansatzes ist DRUM–II flexibler als bisherige Diagnosesysteme. DRUM–II stellt ein breites Spektrum von Diagnosedefinitionen zur Verfügung und unterstützt darüberhinaus allgemeines nicht–monotones Schlußfolgern, z.B. temporales Schlußfolgern. Die Implementierung der Zirkumskription in DRUM–II basiert auf einem neuen korrekten und vollständigen Suchalgorithmus.

Da die Berechnung von Diagnosen eine große Anzahl von Simulationen des Systems unter ähnlichen Annahmen erfordert, werden in Diagnosemaschinen Techniken zur Vermeidung von Mehrfachberechnungen eingesetzt. Bisherige Diagnosemaschinen, die meistens auf Truth Maintainance Techniken basieren, zeichnen während der Berechnung der Diagnosen dynamisch Informationen auf. Im Gegensatz zu diesen Systemen verwendet DRUM–II zur Fokussierung ausschließlich statisches vorkompiliertes Wissen über die Struktur des Systems. Der Platzbedarf dieser vorkompilierten Information ist quadratisch in der Größe des untersuchten Systems, während die in bisherigen Systemen verwendeten Datenstrukturen exponentiellen Platzbedarf aufweisen. Bei Laufzeitmessungen auf Basis der verbreiteten ISCAS–85 Benchmark–Schaltkreise zeigt DRUM–II bei der Diagnose aller Schaltkreise für fast alle Testvektoren höhere Effizienz als bisherige Systeme.

Wir demonstrieren den Einsatz von DRUM–II für die Alarmkorrelation in Mobilfunknetzen. Diese Anwendung stellt ein schwieriges Modellierungsproblem dar, das nur durch Verwendung eines ausdrucksstarken Diagnosekonzeptes, der Spektrum Diagnosen, vollständig zu lösen ist. Wir zeigen, wie Spektrum Diagnosen in DRUM–II durch einen neuen iterativen Algorithmus auf die Zirkumskription zurückgeführt werden.

Schließlich betrachten wir den Prozeßaspekt der Diagnose. Die Berechnung von Diagnosen ist ein dynamischer Prozeß, in dessen Verlauf unterschiedlich abstrakte Systemmodelle und vereinfachende Annahmen verwendet sowie Messungen durchgeführt werden. Wir definieren die erste deklarative Modellierungssprache, die es erlaubt, anwendungsspezifische Diagnoseprozesse zu beschreiben. Darüberhinaus geben wir einen Algorithmus zur Auswertung dieser Prozeßspezifikationen an und demonstrieren seine Anwendung am Beispiel der hierarchischen Schaltkreisdiagnose.

Keywords

Diagnosis, Non-monotonic Reasoning, Artificial Intelligence

Schlagworte

Diagnose, Nicht-monotones Schlußfolgern, Künstliche Intelligenz

Contents

Abbreviations

ATMS	Assumption–based Truth Maintainance System
BSC	Base Station Controller
BTS	Base Station Transceiver
CC	Cross Connect System
CL	Cable Link
DECT	Digital European Cordless Telecommunications
DRUM	Dynamic Revision and Update Machine
GSM	Global System for Mobile Telecommunication
ISCAS	International Symposium on Circuits and Systems
NIHIL	New Implementation of Hyper in Lisp
ML	Microwave Link
MS	Mobile Station
O & M	Operation and Maintenance
OMC	Operation and Maintenance Centre
OSS	Operation Support System
SDH	Synchronous Digital Hierarchy

Chapter 1

Introduction

Diagnosis is one of the central application areas of artificial intelligence. The first diagnostic expert systems developed in the seventies focused mainly on medical diagnosis [BS84, WKA78, Pop82]. Their knowledge bases consisted of empirical rules which deduced diagnoses from symptoms. It was however a hard task to acquire and maintain the large amount of empirical knowledge necessary. Semantical weaknesses of early knowledge representation formalisms added to these problems.

Research in the field of model–based diagnosis has initiated the development of a new generation of diagnostic systems. These systems use declarative logical models to simulate the behavior of an artifact and compute diagnosis by comparing the predictions of this simulation with the actual observed behavior. The complex technical systems, which we encounter today in all areas of everyday life pose challenging applications for these diagnosers. Large technical systems are developed in organized processes and their function is usually documented formally. Model–based diagnosers can exploit these formal descriptions for the simulation of the system's behavior. In this way model–based diagnosis overcomes both the knowledge engineering problems and the semantical problems of rule–based expert systems. They have been successfully used in numerous different application areas.

The diagnosis of digital circuits on the gate level has been used as a proof of concept application since the beginning of work on model–based diagnosis [DH88, Rei87, dKW87]. Later, larger combinatorial circuits have served as a means for comparing the efficiency of diagnostic systems [dK91, RdKS93, FN97]. Recent real–world application of model–based diagnosis of circuits include the diagnosis of VHDL designs [FSW95] as well as the diagnosis of antilock breaking systems in cars [SMS95].

Within the RACE 2 project [SPBL95, dS95] model–based diagnosis was applied to telecommunication networks. The topic of this project was to support on-line maintenance of a network consisting of components from different vendors. Model–based diagnosis has been used successfully in a real time expert system for diagnosing the power distribution network of the Public Utilities Board of Singapore [BNSS93, PN93].

Model–based techniques have also been exploited in medical diagnosis. Gamper and Nejdl [GN97] have used model–based reasoning for the diagnosis of Hepatitis B. Other medical applications of model-based diagnosis include the IDUN system for physiology [Dow92, Dow93], and the KARDIO system [BML89] for model–based interpretation of electrocardiograms.

Perhaps the most exciting recent application of model–based diagnosis is the Livingstone system [WN96b] which will be on board of NASA's first New Millenium spacecraft heading for Saturn in 1998. Livingstone uses a single system description in propositional logic for several reasoning tasks, like fault detection, recovery, reconfiguration and tracking of planner goals. In the New Millennium spacecraft the model–based engine is part of an integrated autonomous architecture, consisting of a planner, which generates abstract plans for achieving high–level goals, an executive, which translates these plans into low–level space craft commands and the model–based diagnosis and reconfiguration component, which tracks the spacecraft's state and proposes the necessary reconfiguration steps.

In [WN96a] Williams and Nayak point out a huge potential potential for model–based systems due to the advent of large autonomous technical systems like chemical plant control systems, building energy systems, autonomous space probes and reconfigurable traffic control systems.

To make model–based diagnosis useful for these complex time–critical applications model–based diagnosis engines must meet the following requirements:

Flexibility: As we pointed out in the spacecraft application, the model–based reasoning engine must be capable of solving many different reasoning tasks based on a uniform description of the system. Thus, a specialized diagnoser is less useful for future applications than a general–purpose reasoner, which can solve a broad range of problems.

Efficiency: Successful model–based engines must be efficient regarding both time and space complexity. Short response times are important because many realistic applications demand reaction in real time. Moreover, diagnosers should be efficient with regards to memory requirements to make them useful for on-board diagnosis.

1.1 Problems Addressed in this Thesis

The central goal of this thesis is to develop a flexible and efficient implementation of model–based diagnosis.

Flexibility means on the one hand that a wide range of diagnostic definitions has to be covered. On the other hand, diagnosis is not the only task, which has to be supported by a model–based reasoning system. Therefore, other forms of reasoning, like temporal reasoning or abductive reasoning should be possible with the engine.

Problem 1. Can we integrate a sufficiently general diagnostic concept in a general framework for reasoning?

In most AI formalisms there is a tradeoff between expressiveness and efficiency. Thus, the reasoning framework must be carefully selected to allow for efficient algorithms. Specialized systems will only be replaced by more general approaches in practice, if the running time of the general system is competitive.

Problem 2. Can we provide an implementation for our reasoner which is competitive with specialized diagnostic systems?

Often it is not enough to have an efficient reasoner. For example, there will often exist multiple explanations for the symptoms observed. Measurements are neccessary to discriminate among competing explanations. Furthermore, there are often different models for one device. In electronics structural, physical and functional models are used on each level of abstraction. A diagnostic system should dynamically decide, which model is best suited for the current situation. Struss [Str92] has pointed out that diagnosis is a dynamical process during which assumptions/models change and actions have to be taken. While the system models used in model–based diagnosis are already declarative, there is great need for declarative descriptions of the diagnostic process.

Problem 3. Can we declaratively specify diagnostic processes, so that the diagnostic system can exploit them efficiently?

A solution to this problem directly contributes to the flexibility and the efficiency of the diagnostic system, because the dynamical choice of assumptions and models increases its flexibility and the use of simplifications and abstractions reduces its running time.

1.2 Solutions Presented in this Thesis

The DRUM–system [Ned93, NG94], the predecessor of DRUM–II, was based on minimal change semantics [Win88, CW91, CW94], which is not fully coherent with the semantics of model–based diagnosis. Due to this semantical problem, DRUM had to use a redundant algorithm, whose performance degraded quickly on large problems.

The first contribution of this thesis is the identification of a logical framework, which directly supports the model–based diagnosis semantics. Several authors have recently chosen monotonic propositional logics as the foundation of their reasoners [WN96b, Lar92, KS96]. Most diagnostic definitions however include some kind of minimization, which is not expressible in monotonic logics. Therefore, we chose circumscription [McC80, McC86], a widely used non–monotonic formalism, as the logical basis of the DRUM–II system. We will show how all relevant diagnostic definitions can be implemented using circumscriptive reasoning. Since circumscription is also the foundation of many formalisms for temporal reasoning, DRUM–II is capable of solving a broader range of reasoning tasks than previous diagnosis engines.

Since its first definition [McC80] several different implementations of circumscription have been proposed [Lif85, Gin89, Prz89]. Most implementations reduce the circumscriptive reasoning to monotonic logics using symbolic manipulations, which are only applicable to theories with certain structural properties. We present a complete implementation of propositional circumscription based on a simple local search procedure. It turns out that the efficiency of our general implementation of circumscription is already sufficient to solve realistic diagnosis problems.

If a technical system is working properly, its output should be a function of its inputs and its environmental parameters. The logical model of this correct behavior is usually a horn theory. We define a technique for automatically exploiting structural information in horn theories, and integrate it into the DRUM–II algorithm. This optimized version of DRUM–II is more efficient than all previous model–based diagnosers in nearly all cases on a widely used benchmark suites of circuit diagnosis problems. Thus, on horn theories we have succeeded in providing a general system for circumscriptive reasoning which handles diagnosis problems more efficiently than specialized diagnosis systems.

While digital circuits are relatively easy to describe in logic, the diagnosis of a cellular network posed a difficult modeling problem. We will show that the widely used consistency–based diagnosis approach is not suitable for this application, because it leads to counter–intuitive models. Through the use of a more general diagnostic concept (spectrum diagnoses) in DRUM–II, we are able to provide a declarative model of this domain.

Propositional circumscription is well suited as a general framework for diagnosis. However, several different reasoning mechanisms with a different focus are used in other areas of AI. To exploit the techniques used in DRUM–II for a wider range of systems, we identify the main ideas underlying the efficiency of DRUM–II and study their integration in a reasoner with different technology. The result is a tableaux–based theorem prover (NIHIL), which is able to solve realistic diagnosis problems. Thus, we combine the expressive power of a first–order logic tableaux calculus with the efficient model–based reasoning techniques in DRUM–II.

Two ingredients are necessary for efficient and flexible diagnosis: the diagnosis system, which interprets the system models and a description of the diagnosis process, which defines suitable system models and actions for the current situation. Although some authors have addressed the problem of integrating process support in diagnostic systems [Str92, BD94] most previous solutions have encoded the diagnostic process in the implementation of the reasoner. In this thesis we present a language for defining the diagnostic process as declaratively as the system description itself. We provide algorithms for efficiently exploiting these process descriptions and give guidelines for designing diagnostic processes.

1.3 Structure of this Thesis

Chapter 2 discusses diagnosis concepts used in model–based diagnosis and their relationships. It also introduces the main ideas underlying the implementations of previous diagnosis engines.

In chapter 3 we introduce the DRUM–II system. First, we describe how DRUM–II implements different variants of propositional circumscription. We demonstrate the use of this implementation by solving some recent temporal reasoning applications. Then, we show how a wide range of diagnosis concepts is handled by DRUM–II.

In chapter 4 we study the diagnosis of combinatorial circuits. We show how circuits can be declaratively described in logics and we compute diagnoses with DRUM–II. Then, we introduce a more efficient version of DRUM–II which exploits the structure of the device under consideration. Finally, we compare the running time of DRUM–II to the running times of previous systems using the ISCAS–85 benchmark suite of combinatorial circuits.

Chapter 5 describes the use of DRUM–II for alarm correlation in cellular networks. Different logical descriptions of a cellular network are developed. The performance of the system on noisy data is assessed.

Chapter 6 shows that the techniques from DRUM–II are also useful for reasoners with different technology. The integration of techniques from DRUM–II into a tableaux–based theorem prover yields a very expressive diagnosis system.

Finally, in chapter 7 we define a language for describing diagnostic processes. We use this language to set up a catalogue of diagnostic strategies for circuit diagnosis. We introduce an efficient algorithm for evaluating these process specifications and evaluate it on a hierarchical circuit problem.

Chapter 2

Model–Based Diagnosis

Early diagnostic expert systems depended largely on a domain expert, whose knowledge about inferring diagnoses from symptoms was expressed by heuristic rules. In contrast to this heuristic approach model–based diagnosis systems compute diagnoses by simulating the behavior of a device and comparing the prediction with the observed behavior. Reiter's seminal paper [Rei87] was the motivation for a broad variety of works on model-based diagnosis, both on the theoretical foundations as well as on efficient algorithms.

While the expressiveness and flexibility of model-based diagnosis is superior to previous approaches, early systems suffered from efficiency problems. Model–based reasoning systems have to partially simulate the device under consideration several times under slightly changing assumptions about the faulty components. We will discuss the most influential model–based diagnosers and their techniques for efficient reasoning.

In the theory of diagnosis several notions of explanation have been studied. While Reiter's definition regards a diagnosis as a set of assumptions consistent with the observations, the abductive diagnosis approach postulates that diagnoses must entail the observations logically. Console and Torasso have defined a spectrum of diagnostic definitions, whose extremes are Reiter's consistency–based diagnosis and abductive diagnosis. The DRUM-II system implements this complete spectrum of definitions to provide a flexible diagnosis concept.

We will conclude this chapter with a critical examination of Console and Torasso's statements about the relationship between abductive and consistency–based reasoning.

2.1 Basic Concepts

In model-based diagnosis a model of the device under consideration is used to predict its behavior. Since the goal of model-based diagnosis is to identify faulty components, this model must explicitly take into account the components of the device and their modes (i.e. if they are working properly or they are faulty). In the logical approach to diagnosis the device is described by a set of logical axioms *SD*, called the system description. To avoid confusion, we will always use the term system description for the model of the device. The term model will from now on only be used in the logical sense, i.e. to denote an interpretation of a set of formulas.

Comp is a set of constants denoting the components of the device. We distinguish at least two modes for each component: A component $c \in Comp$ can be working properly, which is denoted by the proposition $Ok(c)$, or it can be faulty, which is denoted by $Ab(c)$. A set of propositions *Obs* encodes the observed behavior of the system. A diagnosis problem arises, if the observation is inconsistent with the assumption that all components are working properly, i.e. $SD \cup \{Ok(c) | c \in Comp\} \cup Obs \models \bot$.

2.1.1 Consistency–Based Diagnosis

A diagnosis identifies a set of faulty components, so that logical consistency is restored.

Definition 2.1 *Diagnosis*
A Diagnosis *of $SD \cup Obs$ is a set $\Delta \subseteq Comp$ with the property:*

$$SD \cup Obs \cup \{Ab(c) | c \in \Delta\} \cup \{\neg Ab(c) | c \in Comp \backslash \Delta\} \not\models \bot$$

Reiter's original definition of diagnosis appeals to the principle of parsimony: Only a minimal set of components is assumed abnormal. We will follow [dMR92] and refer to this concept as minimal diagnosis.

Definition 2.2 *Minimal Diagnosis*
A diagnosis Δ of $SD \cup Obs$ is called a Minimal Diagnosis *is there is no diagnosis Δ' of $SD \cup Obs$, such that $\Delta' \subset \Delta$.*

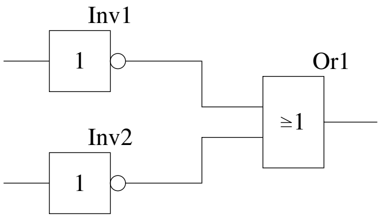


Figure 2.1: A simple digital circuit

Example 2.3 Consider the simple digital circuit in figure 2.1 consisting of an or–gate ($Or1$) and two inverters ($Inv1$ and $Inv2$). The system description *SD* is given by the following formulas.

$$\begin{array}{rl}
(R_1) & \forall c\, Type(c, Or) \land \neg Ab(c) \\
 & \quad \rightarrow (High(c, O) \leftrightarrow High(c, I1) \lor High(c, I2)) \\
(R_2) & \forall c\, Type(c, Inv) \land \neg Ab(c) \rightarrow (High(c, O) \leftrightarrow \neg High(c, I)) \\
(R_3) & \forall c_1, p_1, c_2, p_2\, Conn(c_1, p_2, c_2, p_2) \rightarrow (High(c_1, p_1) \leftrightarrow High(c_2, p_2)) \\
(F_1 \Leftrightarrow F_3) & Type(Or1, Or),\ Type(Inv1, Inv),\ Type(Inv2, Inv) \\
(F_4 \Leftrightarrow F_5) & Conn(Inv1, O, Or1, I1),\ Conn(Inv2, O, Or1, I2)
\end{array}$$

In this system description we have separated general knowledge about the components involved (R_1, R_2, and R_3) from the knowledge about the topology of the actual device (F_1, …, F_5). This separation is typical for model–based diagnosis and keeps system descriptions maintainable. The first rule R_1 describes the behavior of a component c, which is an Or-gate ($Type(c, Or)$). If an Or-gate is behaving according to its specification, the output has high voltage, if and only if one of its inputs has high voltage. R_2 denotes that a correctly functioning inverter inverts its input value. R_3 states that two connected ports have the same voltage. The facts F_1, F_2, and F_3 introduce the components of the given circuit while the facts F_4 and F_5 describe their connections. We observe that both inputs of the circuit have low voltage and the output also has low voltage.

$$Obs = \{\neg High(Inv1, I), \neg High(Inv2, I), \neg High(Or1, O)\}$$

Under these observations two minimal diagnoses exist:

$$\begin{array}{rcl}
\Delta_1 & = & \{Ab(Or1)\} \\
\Delta_2 & = & \{Ab(Inv1), Ab(Inv2)\}
\end{array}$$

The low voltage at the output of the or-gate can only be explained if either the or-gate itself is behaving abnormally or both of its inputs are low and thus both inverters are faulty. Of course, $\{Ab(Or1), Ab(Inv1), Ab(Inv2)\}$ is also a diagnosis but it is not minimal and therefore not considered by Reiter's definition. #

From definition 2.1 it is obvious that computing diagnoses requires deciding satisfiability for the language used. Thus, using full first–order logic as the underlying language will inevitably lead to an incomplete diagnosis algorithm. We therefore use a restricted language throughout this book, which does not include function symbols. This language is formally defined in section 3.4.1.

One motivation for computing only the minimal diagnoses is that minimal diagnoses are often sufficient to characterize the space of all diagnoses: Usually, every superset of a minimal diagnosis is also a diagnosis. De Kleer, Mackworth and Reiter have critically examined this intuition in [dMR92] and qualified it as summarized in the following theorem.

Theorem 2.4 *Let $(SD, Comp, Obs)$ be a representation of a diagnosis problem, such that Ab occurs only positively in the clause form of SD.*

For every set of Δ components: If $\Delta \supset \Delta'$ for a minimal diagnosis Δ', then Δ is itself a diagnosis.

Extending our previous example we can easily construct a case where a superset of a diagnosis is not itself a diagnosis.

Example 2.5 Let us extend the system description from example 2.3 by

$$R_4 : \forall c\ Type(c, Or) \wedge Ab(c) \rightarrow High(c, O)$$

The clause corresponding to this formula is $\{\neg Type(c, Or), \neg Ab(c), High(c, O)\}$, in which Ab occurs negatively so that the preconditions of theorem 2.4 are not satisfied. In fact, for the observation from example 2.3 we obtain only one diagnosis $\Delta_2 = \{Ab(Inv1), Ab(Inv2)\}$. $Comp \supset \Delta_2$ is no diagnosis. #

2.1.2 Kernel Diagnoses

In diagnosis vocabulary a formula like R_4 in above example, which describes the behavior of a faulty component, is called a fault model. De Kleer, Mackworth and Reiter have coined the concept of a kernel diagnosis [dMR92] to characterize the diagnosis space in the presence of fault models. They define a diagnosis as a conjunction of *Ab*-literals (positive or negative), which is consistent with *SD* and *Obs*.

Definition 2.6 *Diagnosis (following [dMR92])*
A conjunction D of Ab-literals is a Diagnosis *of $(SD, Comp, Obs)$, iff*

$$SD \cup Obs \cup D \not\models \bot$$

While Reiter has used set inclusion as minimality criterion, de Kleer, Mackworth, and Reiter use the concept of *Covering*.

Definition 2.7 *covers*
A conjunction C of literals covers a conjunction D of literals, iff every literal in C occurs in D.

A *Partial Diagnosis* helps characterizing the space of all diagnoses, because all conjunctions of *Ab*-literals, which cover it, are diagnoses.

Definition 2.8 *Partial Diagnosis*
A Partial Diagnosis *is a diagnosis, such that all conjunctions of Ab-literals covered by it are diagnoses.*

The partial diagnoses which are minimal wrt. covering are called *Kernel Diagnoses*.

Definition 2.9 *Kernel Diagnosis*
A Partial Diagnosis is called a Kernel Diagnosis*, if no other diagnosis covers it.*

Although kernel diagnoses thoroughly characterize the space of all diagnoses this concept had no strong influence on diagnostic systems. The reason is that computing kernel diagnoses is a very expensive task and the set of all kernel diagnoses is too large for complex devices.

2.1.3 Reducing the Number of Diagnoses

For large systems even the set of all minimal diagnosis is too big to allow efficient computation. Furthermore, the user of a diagnostic system wants a small set of diagnoses, which directs him to the faulty components. In this section we discuss diagnosis concepts which omit less likely minimal diagnoses. The best way to discriminate among diagnoses is to execute measurements [dKW87, dK90b]. However, measurements require taking possibly costly actions in the real world.

A widely used approach to discriminate further among the minimal diagnoses is to rank them according to their prior probability. Assuming statistical independence among the possible failures of a system (which may be a too strong assumption for some domains), the prior probability that a given diagnosis is correct is ([dK90b])

$$P(\Delta) = \prod_{c \in \Delta} p_c \cdot \prod_{c \in Comp \setminus \Delta} (1 \Leftrightarrow p_c), \tag{2.1}$$

where p_c is the probability, that component c fails.

Definition 2.10 *Maximally Probable Diagnosis*
For a diagnosis Δ let $P(\Delta)$ be defined by equation 2.1. A diagnosis Δ is called a Maximally Probable Diagnosis*, iff there is no diagnosis Δ' such that $P(\Delta') > P(\Delta)$.*

If all probabilities p_c are equal (or assumed to be equal due to lack of information), i.e. $p_c = p$ for all $c \in Comp$ we have

$$P(\Delta) = p^{|\Delta|} \cdot (1 \Leftrightarrow p)^{|Comp| - |\Delta|} \tag{2.2}$$

In this case, the probability of a diagnosis depends only on its cardinality and we can rank the diagnoses simply by their cardinality.

Definition 2.11 *Minimal Cardinality Diagnosis*
For a diagnosis Δ let $|\Delta|$ denote the number of components in Δ.

A diagnosis Δ is called minimal cardinality diagnosis, iff there is no diagnosis Δ' so that $|\Delta'| < |\Delta|$.

Restrictions on the cardinality of diagnoses are very common. Many (non–model–based) diagnosis systems have implicit restrictions on the cardinality of diagnoses: Often they can only compute single fault diagnoses, i.e. diagnoses with cardinality 1.

2.2 Computing Diagnoses

An obvious way to compute diagnoses following definition 2.2 is to enumerate diagnosis candidates Δ (i.e. sets of components considered faulty) and check if $SD \cup Obs \cup \{\neg Ab(c) | c \in Comp \backslash \Delta\} \cup \{Ab(c) | c \in \Delta\}$ is logically consistent. If so, Δ is a diagnosis.

Reiter's Framework. Reiter motivates his diagnosis algorithm [Rei87] by the statement that a generate and test approach is too inefficient for computing diagnoses with large cardinality because a large number of candidates has to be checked. Instead of enumerating candidates Reiter takes up an idea proposed by de Kleer [dK76] and uses conflict recognition as a preliminary stage to candidate generation.

Definition 2.12 *Conflict Set.*
A Conflict Set *for $(SD, Comp, Obs)$ is a set $\{c_1, \ldots, c_n\} \subseteq Comp$ such that*

$$SD \cup Obs \cup \{\neg Ab(c_1), \ldots, \neg Ab(c_n)\} \models \bot$$

Conflict sets can be generated using a theorem prover by computing a refutation for $SD \cup Obs \cup \{\neg Ab(c) | c \in Comp\}$. Every set of Ab-literals used in such a refutation is a conflict (set). A candidate has to take every conflict into account. This is formalized by the notion of a *Hitting Set*.

Definition 2.13 *Hitting Set*
A Hitting Set *for a collection $\mathcal{C}$ of sets is a set $H \subseteq \bigcup_{S \in C} S$ such that $H \cap S \neq \emptyset$ for every $S \in \mathcal{C}$.*

A hitting set is called minimal, if it is the smallest set with the above property.

Reiter proves that the minimal diagnoses correspond to the minimal hitting sets of the set of conflicts. To compute hitting sets systematically, Reiter uses a data structure called hitting set tree. In a hitting set tree internal nodes are labeled by conflicts. For every component in the conflict, an outgoing edge labeled with this component is added to the node as shown in figure 2.2. The hitting sets correspond to the sets of edge labels on a path. A branch is extended as long as there is a conflict not yet covered by the set of literals on the branch.

Since only the minimal hitting sets correspond to minimal diagnoses, Reiter proposes techniques for eliminating nodes from the HS-tree, which provably do not lead to minimal hitting sets. All nodes, which correspond to a superset of an already found hitting set are eliminated. Reiter defined an additional optimization concerning the

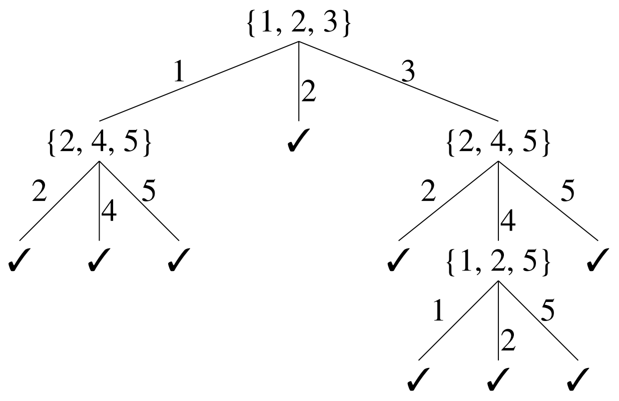


Figure 2.2: Hitting Set Tree for $\{\{1,2,3\},\{2,4,5\},\{1,2,5\}\}$. The branches $(1,2),(3,2),(3,4,1),(3,4,2)$, and $(3,4,5)$ correspond to non–minimal hitting set and could be pruned.

nodes labeled by non-minimal conflicts. This technique however contained an error which was later corrected in the HS-DAG algorithm by Greiner, Smith, and Wilkerson in [GSW89].

It should be noted that the techniques in the HS-tree/HS-DAG lead to a considerable overhead due to the large number of costly subset checks involved. We know of no experimental or theoretical evaluation of the benefits of these methods. Moreover, it is sometimes seen as an advantage of Reiter's algorithm, that non-minimal conflicts can be used [GSW89]. Since however a relatively complex algorithm is needed for deleting them from the HS–DAG (again using a large number of subset checks), this advantage is questionable.

GDE. In their GDE system [dKW87] de Kleer and Williams also divide the computation of diagnoses into the phases conflict generation and candidate generation. The candidate generator in GDE maintains the set of currently minimal candidates and updates it accordingly for each new minimal conflict. In contrast to Reiter, who makes only some general statements on conflict generation, de Kleer and Williams carefully design an efficient conflict recognition algorithm. This algorithm is based on the notion of an *Environment*. An environment is a set of components considered OK. The system description together with an environment causes a set of predictions. If any of the predictions contradicts the observations, the environment is a conflict. GDE finds all minimal conflicts by systematically enumerating environments starting with the empty environment and moving up the subset/superset lattice of components successively.

Since most values do not depend on the complete environment but rather only on a subset thereof, a large number of predictions (and thus inference operations) would be repeated for the different environments, if a straightforward inference architecture were used. De Kleer and Williams therefore use truth maintenance techniques to avoid such recomputations. With each value a minimal supporting environment is stored. The value can be assumed in every environment which includes its supporting environment as a subset.

Sherlock. The original GDE was limited to descriptions of the correct behavior only. In [dKW89] de Kleer and Williams introduced fault models into GDE and called the resulting system Sherlock. They found that the additional combinatorics of using fault models made the system too inefficient to solve even small problems. To deal with this complexity de Kleer and Williams abandon the idea of computing all minimal diagnoses and introduce the notion of *Leading Diagnoses*. Leading diagnoses are the most probable diagnoses of the system. Heuristic criteria define the probability up to which a diagnosis is considered a leading diagnosis. The set of leading diagnoses is usually small (e.g. five diagnoses).

AAAI91. The ideas and algorithms concerning the computation of diagnoses presented so far were either of theoretical nature [Rei87] or checked only on small examples [dKW87], [dKW89]. When de Kleer started working on large combinatorial benchmark circuits [Isc85] and large circuits consisting of cascaded elements he found that both GDE and Sherlock were too inefficient to handle large examples. In [dK91] de Kleer shows that all three phases of computation (prediction, conflict generation, candidate generation) in GDE and Sherlock exhibit combinatorial explosion. De Kleer's next diagnosis engine reduces the combinatorial explosion by focusing the diagnosis engine on the leading diagnoses. On the one hand this is achieved by incorporating the estimation of probabilities into an incremental candidate generator, which returns only the single most probable candidate not considered so far. On the other hand the truth maintenance system used in GDE and Sherlock (ATMS) is replaced by a more efficient technology (HTMS) which reduces the combinatorial explosion of conflicts and predictions.

IMPLODE. Even the system from [dK91] did not solve all combinatorial benchmark circuits from the ISCAS-85 suite. In [RdKS93] Raiman, de Kleer and Saraswat introduce the IMPLODE system, which reduces the number of environments and conflicts drastically by exploiting the concept of criticality. The *Critical Environment* for a literal l is the intersection of all environments, under which l is entailed. Let us denote the critical environment for a literal l by $CE(l)$. The basic idea behind critical reasoning is the (unsound) abstraction $\left(\bigwedge_{x \in CE(l)} x\right) \to l$. When this abstraction is applicable, it focuses the reasoning dramatically, because all the alternative environments, under which l is entailed, do not have to be considered.

Example 2.14 Let us assume that the atom $High(C5,O)$ is entailed under the minimal environments $\{Ok(C1), Ok(C2)\}$ and $\{Ok(C1), Ok(C4)\}$. The critical environment for

$High(C5,O)$ is the intersection of these environments, i.e. $\{Ok(C1)\}$. The abstraction $Ok(C1) \rightarrow High(C5,O)$ is introduced. #

Although the abstractions computed in critical reasoning are unsound in general, Raiman, de Kleer and Saraswat show that they are valid under certain conditions. For example, if we are interested in single faults, the above abstraction is consistent: If $C1$ is assumed abnormal, $High(C5,O)$ will not be assumed. If any other component is considered abnormal, then either the environment $\{Ok(C1), Ok(C4)\}$ or $\{Ok(C1), Ok(C2)\}$ remains intact and the value can be deduced. Set covering techniques, which however have not been considered in depth in [RdKS93] can be used to generalize the applicability of these abstractions in a multiple fault scenario.

2.3 A Spectrum of Diagnosis Definitions

Diagnosis following definition 2.1 is often referred to as consistency-based diagnosis. This is a natural concept, if the system description only describes the correct behavior of the components. If we additionally have fault models, we may want a stronger diagnosis concept, which does not only postulate consistency but explanation (in the sense of logical entailment) of the observed misbehavior. To formalize such a stronger notion of diagnosis, let us first introduce some additional notation. From now on, we assume that each component c has one correct mode Ok and several fault modes $Fm_{c1}, \ldots Fm_{cn_c}$. By $Mode(c,m)$ we denote that component c is in mode m. We define

$$\begin{array}{l} \forall c\ (Ok(c) \leftrightarrow Mode(c, Ok)) \\ \forall c\ (Ab(c) \leftrightarrow \exists m\ (Mode(c,m) \wedge m \neq Ok)) \end{array}$$

so that we can still use our previous definitions. Furthermore we postulate, that every component is in exactly one mode at a given time point, i.e.

$$\forall c \exists m\ (Mode(c,m) \wedge \forall m'\ (m' \neq m \rightarrow \neg Mode(c,m')))$$

Further we partition the observations Obs into a set Obs_{In} of parameters (input observations) and a set Obs_{Out} of output observations, such that $Obs = Obs_{In} \dot{\cup} Obs_{Out}$. In the presence of multiple fault modes, a diagnosis can no longer be denoted as a set of components. We now need the concept of a *Mode Assignment*.

Definition 2.15 *Mode Assignment.*
A set D of atoms is called a Mode Assignment*, iff*

1. *D contains exactly one atom of the form $Mode(c,m)$ for every component $c \in Comp$ and*

2. *D contains no other atoms.*

When working with mode assignments, we still want to minimize abnormality. Therefore, we introduce an ordering $\leq^{Ab}$ on the mode assignments, similar to the ordering on models.

Definition 2.16 $\leq^{Ab}$
For mode assignments D_1 and D_2 we define

$$D_1 \leq^{Ab} D_2, \textit{ iff } \{c|Mode(c,Ab) \in D_1\} \subseteq \{c|Mode(c,Ab) \in D_2\}$$

An *Abductive Diagnosis* is a mode assignment consistent with system description and observations, which entails all output observations.

Definition 2.17 *Abductive Diagnosis.*
Let $(SD, Comp, Obs = Obs_{In} \dot{\cup} Obs_{Out})$ be a diagnostic problem. A mode assignment M is called an Abductive Diagnosis*, iff*

1. $SD \cup Obs \cup M \not\models \perp$ *and*

2. $SD \cup Obs_{In} \cup M \models Obs_{Out}$

In [CT91] Console and Torasso point out that a spectrum of diagnostic definitions exists between the extremes consistency–based diagnosis and abductive diagnosis. The definitions in the spectrum differ in the size of the subset $Obs^+ \subseteq Obs_{Out}$ of observations, which have to be explained. The extremes are $Obs^+ = \emptyset$ for consistency–based diagnosis and $Obs^+ = Obs_{Out}$ for abductive diagnosis[1]. Following Console we can provide a general definition of a diagnosis problem which includes the specification of Obs^+ and a diagnosis concept which covers the whole spectrum.

Definition 2.18 *Diagnosis Problem, Spectrum Diagnosis.*
A Diagnosis Problem *is given by $(SD, Comp, Obs = Obs_{In} \dot{\cup} Obs_{Out}, Obs^+)$ where $Obs^+ \subseteq Obs_{Out}$.*

A mode assignment M is a Spectrum Diagnosis*, iff*

1. $SD \cup Obs \cup M \not\models \perp$ *and*

2. $SD \cup Obs_{In} \cup M \models Obs^+$

By implementing spectrum diagnosis, the DRUM–II system covers a broad range of diagnostic definitions and applications. Additionally, DRUM–II allows to restrict the set of all minimal spectrum diagnoses further by cardinality or probability as discussed in section 2.1.3.

[1] Console defines another dimension of the spectrum by dividing the observations into normal and abnormal observations

2.4 On the Role of Abductive Diagnosis

Console, Theseider Dupré and Torasso have formally studied the relationship between abductive and consistency–based reasoning in [CDT91]. They show that abductive explanations are strictly equivalent to the models of the completed (via predicate completion) theory. With a more informal and domain–oriented argumentation, Console and Torasso present similar equivalences between abductive and consistency–based diagnosis in [CT91].

We however argue that the completions presented in [CT91] and [CDT91] are often not applicable to system descriptions of technical devices. Complex technical systems usually consist of a large number of standard components connected in a network structure. The system description usually formalizes propagation of certain properties through this component network. Examples are the water pressure in a system of pipes and valves, the current in a switching network, or the presence of messages in a communication network. To model such propagations we often use rule–like axioms. In a communication network, the propagation of a message through the network can be formalized by a predicate $P(x,y)$, denoting that the message sent by component y has reached component x.

$$\forall x \forall y \forall z \; P(x,x) \wedge (P(x,z) \wedge Conn(x,y) \to P(y,z)) \tag{2.3}$$

The above axiom describes the propagation of status messages through a network: Each component sends a status message ($P(x,x)$), and each component forwards status messages to the connected components. In our telecommunication application in chapter 5, we use similar but more complex axioms to describe the forwarding of status messages through the network. It is interesting to see that already the simple axiom in formula 2.3 cannot be completed in first order logic. To see this, note that the completion of axiom 2.3 is exactly the transitive closure $Conn^*$ of the relation $Conn$: A component x forwards a status message to all components directly or indirectly connected to it. Thus, we have $Conn^*(x,y) \Leftrightarrow P(y,x)$, i.e. the component x is connected to the component y, iff y has received the status signal of x. However, the transitive closure cannot be formalized in first–order logic.

Proposition 2.19 *The transitive closure cannot be formalized by a set of first–order formulas.*

Proof: For this proof we will denote interpretations as pairs (M,β), consisting of a frame M and an assignment β. We will show that there is no first order theory $\mathcal{T}$ such that $MOD(\mathcal{T})$ (the set of all models of $\mathcal{T}$) has the property:

For every $M \in MOD(\mathcal{T})$: $M|R^*|$ is the transitive closure of $M|R|$[2].

[2] R^* is treated as a predicate constant

The proof is by contradiction. Suppose $\mathcal{T}$ is such a theory. Let us define a set of frames $\{M^n | n \in \mathbf{N}\}$. The domain of M^k is the set $\{1, \ldots, k\}$. Further we define

$$\begin{array}{rcl} M^k|R| & := & \{(i, i+1) | 1 \leq i < k\} \\ M^k|R^*| & := & \{(i, j) | 1 \leq i < j \leq k\} \end{array}$$

Then, obviously $M^k|R^*|$ is the transitive closure of $M^k|R|$. Let

$$\phi_k := x \neq y \wedge R^*(x,y) \wedge \neg(\exists x_1 \ldots \exists x_k \; x_1 = x \wedge x_k = y \wedge R(x_1, x_2) \wedge \ldots \wedge R(x_{k-1}, x_k)).$$

The formula ϕ_k expresses that R^* holds for x and y but y is not reached by k times following the relation R. Let

$$\mathcal{T}' := \mathcal{T} \cup \{\phi_k | k \geq 2\}.$$

We will show that $\mathcal{T}'$ is satisfiable by exploiting the compactness theorem of first order logic. Thus, we have to show that every finite subset $\mathcal{T}_\subset$ of $\mathcal{T}'$ is satisfiable. For a given finite subset $\mathcal{T}_\subset$ there is a largest number j such that ϕ_j is contained in $\mathcal{T}_\subset$ (otherwise $\mathcal{T}_\subset$ would be infinite). For this j we have $\mathcal{T}_\subset \subseteq \underbrace{\mathcal{T} \cup \{\phi_k | 2 \leq k \leq j\}}_{\mathcal{T}_j}$.

The interpretation (M^{j+1}, β) with $\beta(x) = 1$ and $\beta(y) = j+1$ is obviously a model of $\mathcal{T}_j$ because M^{j+1} interprets R^* as the transitive closure of R and $j+1$ applications of R are needed to get from 1 to $j+1$, so that all ϕ_k for $2 \leq k \leq j$ are satisfied. Thus $\mathcal{T}_j$ has a model, which is also a model of $\mathcal{T}_\subset$, because $\mathcal{T}_\subset \subseteq \mathcal{T}_j$.

Since every finite subset of $\mathcal{T}'$ has a model, by compactness theorem, we infer that $\mathcal{T}'$ itself has a model. However, this model does obviously not interpret R^* as the transitive closure of R, because for some x, y we have $R^*(x,y)$ but y cannot be reached from x by finitely many applications of R. Since this model of $\mathcal{T}'$ is also a model of $\mathcal{T}$ (because $\mathcal{T} \subseteq \mathcal{T}'$), we have found a contradiction to the assumption that all models of $\mathcal{T}$ interpret R^* as the transitive closure of R.

Q.E.D.

The reduction of our propagation problem to the well–known result for the transitive closure was only possible, because P has two arguments, one of which identifies the sender of the message. However, we can generalize our result further. Consider the following propagation axiom:

$$\forall x \forall y \; (C_P(x) \rightarrow P(x)) \wedge (P(x) \wedge Conn(x,y) \rightarrow P(y)).$$

This is in some sense the simplest possible form of propagation. A property is present at x if x is the creator or cause of the property P ($C_P(x)$). Furthermore, properties are propagated over connections.

We can modify the proof of proposition 2.19 to show that even this simple type of propagation cannot be described in first–order logic.

Proposition 2.20 *The completion of formula 2.4 cannot be formalized by a set of first–order formulas.*

Proof: We will show that there is no first order theory $\mathcal{T}$ such that $MOD(\mathcal{T})$ has the property:

$$(*)\left\{\begin{array}{l}\text{For every } M \in MOD(\mathcal{T})\text{: } x \text{ is in } M|P|\text{, iff } x \text{ is in } M|C_P| \text{ or there exists an } y \text{ such} \\ \text{that } y \text{ is in } M|C_P| \text{ and } (y,x) \text{ is in the transitive closure of } M|Conn|.\end{array}\right.$$

The proof is by contradiction. Suppose $\mathcal{T}$ is such a theory. Let us define a set of frames $\{M^n|n \in \mathbf{N}\}$. The domain of M^k is the set $\{1,\ldots,k\}$. Further we define

$$\begin{array}{rcl} M^k|P| & := & \{1,\ldots,k\} \\ M^k|C_P| & := & \{1\} \\ M^k|Conn| & := & \{(i,i+1)|1 \leq i < k\} \end{array}$$

Then, obviously M^k satisfies $(*)$. Now let

$$\phi_k := P(x) \wedge \neg(\exists x_1 \ldots \exists x_k \; C_P(x_1) \wedge x_k = x \wedge R(x_1,x_2) \wedge \ldots \wedge R(x_{k-1},x_k)).$$

The formula ϕ_k expresses that P holds for x but x is not reached by k times following the relation R from a point, where C_P holds. Let

$$\mathcal{T}' := \mathcal{T} \cup \{\phi_k|k \geq 2\}.$$

Again, we have to show that every finite subset $\mathcal{T}_\subset$ of $\mathcal{T}'$ is satisfiable. For a given finite subset $\mathcal{T}_\subset$ there is a largest number j such that ϕ_j is contained in $\mathcal{T}_\subset$. For this j we have $\mathcal{T}_\subset \subseteq \underbrace{\mathcal{T} \cup \{\phi_k|2 \leq k \leq j\}}_{\mathcal{T}_j}$.

The interpretation (M^{j+1},β) with $\beta(x) = j+1$ and is obviously a model of $\mathcal{T}_j$ because the frame M^{j+1} satisfies $(*)$ and $j+1$ applications of R are needed to get from 1 to $j+1$, so that all ϕ_k for $2 \leq k \leq j$ are satisfied. Thus $\mathcal{T}_j$ has a model, which is also a model of $\mathcal{T}_\subset$, because $\mathcal{T}_\subset \subseteq \mathcal{T}_j$.

Since every finite subset of $\mathcal{T}'$ has a model, by compactness theorem, we infer that $\mathcal{T}'$ itself has a model. However, this model does not satisfy $(*)$, because for some x, $P(x)$ holds, but it is not connected to a point where $C_P(x)$ holds. Since this model of $\mathcal{T}'$ is also a model of $\mathcal{T}$ (because $\mathcal{T} \subseteq \mathcal{T}'$), we have found a contradiction to the assumption that all models of $\mathcal{T}$ satisfy $(*)$.

Q.E.D.

We conclude that abduction cannot be substituted by first–order deduction in the diagnosis of systems with a network structure. Thus, we need either higher–order logic, non–monotonic logic or an implementation of abduction for the reasoning. Due to the lack of practical reasoning methods for higher order logics, this alternative seems to be inadequate. In this thesis we therefore exploit the other alternatives, non–monotonic logics and the direct implementation of abduction.

2.5 Discussion

Model–based diagnosis allows to exploit declarative logical system descriptions for the diagnosis task. Most work has focused on consistency–based diagnosis and its variants. Several years of research were necessary to create consistency based diagnosis engines with satisfactory performance. Most of these systems are based on conflict detection and candidate generation. Because of the continuing efficiency problems of conflict–based engines, Reiter's initial claim, that combinatorial explosion of the running time can be avoided by conflict generation techniques, seems questionable. It is therefore worthwhile studying alternative algorithms, as used by the DRUM–II system.

Console and Torasso have extended the model–based diagnosis paradigm by introducing a spectrum of diagnostic definitions. The have shown how to reduce the definitions in their spectrum to consistency–based reasoning. However, we have shown that these reductions, which work well for some examples, do not apply to the diagnosis of systems with network structure described in first order logic.

The following chapters of this thesis are concerned with an alternative efficient implementation of model–based diagnosis which is able to handle the whole spectrum of diagnostic definitions for the diagnosis of large systems with network structure.

Chapter 3

The DRUM-II Framework

As we discussed in the section 2.2 most previous systems for model–based diagnosis are based on conflict recognition and candidate generation. Despite their application–specific algorithms, they suffer from combinatorial explosion of internal data structures during the computation of diagnoses. Thus, it is worthwhile to consider alternatives to the conflict–based algorithms motivated in Reiter's work. Furthermore, there is a trend in recent logic–based AI research to move away from specialized logics/algorithms and solve problems using general logical inference engines instead. Examples for this development are the trend to solve planning/scheduling problems by reducing them to propositional logic [KS96] and the reduction of test pattern generation problems to propositional logic [Lar92].

DRUM–II also uses a generic logical engine to compute diagnoses. However, the underlying logics has to be more expressive than propositional logic, because most diagnosis concepts appeal to the principle of parsimony: We want to assume only a minimal set of faulty components. To account for the principle of parsimony the underlying logic has to include a means of minimizing the extension of the *Ab*–predicate. Several non–monotonic logics have been proposed for this purpose [Rei80, McC80]. We have chosen Circumscription, because its semantics corresponds directly to model–based diagnosis and it is the basis of many temporal reasoning formalisms.

We show that consistency–based diagnosis corresponds directly to the computation of the minimal models which characterize the circumscription semantically. The more expressive spectrum diagnosis concept is implemented by iterating minimal model computation (for the consistency–based part) and deciding entailment (for the abductive part). Both tasks are solved with our implementation of circumscription. By implementing circumscription efficiently DRUM–II provides on the one hand a very efficient system for model–based diagnosis and on the other a flexible framework for non–monotonic reasoning in general. This chapter extends and generalizes our previous results in [FN96a, FN96b].

3.1 Introduction

Circumscription [McC80, McC86] is one of the most popular formalisms for non–monotonic reasoning. It is used to provide a semantics and reasoning method for statements like the following:

> Normally, basket ball players are tall.

We can try to formalize this statement in first order logic by the sentence

$$\forall x\, BasketBallPlayer(x) \wedge \neg Ab(Height, x) \rightarrow Tall(x).$$

This sentence reads: A basketball player, who is not abnormal with respect to his height, is tall. However, monotonic logic does not capture the intended meaning of this sentence very well. Suppose, we know that *Tom* is a basketball player. Then, monotonic logic would not allow us to conclude that *Tom* is tall, because he could also be abnormal regarding size. Our intention is however: If we are not forced to assume that *Tom* is abnormal wrt. height, we assume that he is tall.

Circumscription helps in this situation by minimizing the extension of certain predicates, like the predicate *Ab* in the above example. Formally, the *circumscription of a theory $\mathcal{T}$ in a predicate P* adds a second order sentence (called circumscription axiom) to $\mathcal{T}$, which postulates that the extension of P is minimal, i.e. no superfluous atoms $P(x)$ will be assumed. If we circumscribe the theory

$$\begin{array}{ll} \mathcal{T}: & \forall x\, BasketBallPlayer(x) \wedge \neg Ab(Height, x) \rightarrow Tall(x). \\ & BasketBallPlayer(Tom) \end{array}$$

in the predicate *Ab*, we can infer that *Tom* is tall, because is it consistent to assume that he is not abnormal regarding height. To decide, if a formula φ follows from the theory $\mathcal{T}$ circumscribed in a predicate P, most approaches compute the circumscription axiom and (if possible) reduce it to a first order sentence [Lif85]. Then monotonic logic is used to do the proof. Unfortunately, in many cases the circumscription axiom cannot be reduced to a first order sentence. Since there are no efficient reasoning methods for second order logic in general, the scope of this reduction method is limited.

Therefore, some authors have investigated the so–called semantical characterization of circumscription. This approach defines the semantics of circumscription by minimality criteria for models. This characterization has been used by Ginsberg to create an ATMS–based circumscriptive theorem prover [Gin89].

In this chapter we present a more direct approach for exploiting this characterization, which uses model–based reasoning techniques to compute minimal models of a given theory. By dividing the minimal models into equivalence classes we only need to compute a representative subset of the minimal models. These few models are then used by a filtering function to decide, if a given formula φ follows from the circumscription. The proof is done by showing that there is no minimal model, in which $\neg\varphi$

holds. This refutation method was already used by Przymusinski [Prz89], whose query answering method is based on resolution.

We provide efficient algorithms for our approach which make use of the model–based reasoning techniques introduced by Chou and Winslett [CW94]. We show that these algorithms are sound and complete for fixed–domain theories [Luk90]. Our approach avoids problems with the symbolic manipulation and reduction of the circumscription axiom and provides an efficient way to handle current applications. We underline this by implementing current formalisms for reasoning about action and change using our system.

3.2 The Model–based Approach

3.2.1 Definition of Minimal Models

For a model M and a predicate symbol K we write $M|K|$ to denote the extension of K in M. In [McC86], McCarthy defines Formula Circumscription, which allows the extensions of certain predicates to vary during minimization. We will first consider the special case of circumscribing a theory $\mathcal{T}$ in a tuple $\bar{P}$ of predicates, while varying the extensions of all other predicates and generalize this to other variants of circumscription in section 3.3. The semantics for this case of circumscription (usually referred to as parallel circumscription) can be characterized by the $<^{\bar{P}}$ minimal models of $\mathcal{T}$.

Definition 3.1 *Let $\bar{P} = (P_1, \ldots, P_n)$ be a tuple of predicates. Let M_1 and M_2 be models. $M_1 \leq^{\bar{P}} M_2$, iff*

1. *M_1 and M_2 have the same domain and agree in the interpretation of the constant symbols.*
2. *For all $i \in \{1, \ldots, n\}$: $M_1|P_i| \subseteq M_2|P_i|$.*

*A model M is called a $<^{\bar{P}}$–*minimal model *of the theory $\mathcal{T}$, iff there is no model M' of $\mathcal{T}$, such that $M' <^{\bar{P}} M$. We will denote the set of $<^{\bar{P}}$–minimal models of $\mathcal{T}$ by $MOD^{\bar{P}}(\mathcal{T})$.*

$$MOD^{\bar{P}}(\mathcal{T}) := \{M | M \models \mathcal{T} \wedge (\nexists M' : M' <^{\bar{P}} M \wedge M' \models \mathcal{T})\}$$

The following theorem found by Lifschitz [Lif85] shows the connection between circumscription and the set of $<^{\bar{P}}$–minimal models.

Theorem 3.2 *The formula φ follows from the circumscription of $\mathcal{T}$ in $\bar{P}$, while varying all other predicates, iff φ holds in all $<^{\bar{P}}$–minimal models of $\mathcal{T}$.*

Thus, if we knew all $<^{\bar{P}}$–minimal models of a given theory, we could answer queries concerning the circumscribed theory just by looking at the models. However, computing and storing all $<^{\bar{P}}$–minimal models is usually very inefficient. We will describe a method for answering queries which needs only a few models, by introducing an equivalence relation on models.

3.2.2 Computing Minimal Models

Two models are defined to be equivalent wrt. $\bar{P}$, if the extensions of all predicate constants in $\bar{P}$ are the same in both models.

Definition 3.3 *Let $\bar{P} = (P_1, \ldots, P_n)$ be a tuple of predicate constants. Two models M and M' are $\bar{P}$–*equivalent *(denoted by $M \sim^{\bar{P}} M'$), iff for all $i \in \{1, \ldots, n\}$: $M|P_i| = M'|P_i|$. By $[M]$, we denote the set of all models $\bar{P}$–equivalent to M, i.e. $[M] = \{M' | M \sim^{\bar{P}} M'\}$. For a set $\mathcal{M}$ of models we define $\mathcal{M}/\sim^{\bar{P}} := \{[M] | M \in \mathcal{M}\}$.*

Each equivalence class is represented by (stored as) some model $\tilde{M} \in [M]$. Thus we work on finite sets of models representing $MOD^{\bar{P}}(\mathcal{T})/\sim^{\bar{P}}$.

Definition 3.4 *A set $\mathcal{M}$ of models is a* Transversal *of $MOD^{\bar{P}}(\mathcal{T})/\sim^{\bar{P}}$, iff it contains exactly one model out of every equivalence class in $MOD^{\bar{P}}(\mathcal{T})/\sim^{\bar{P}}$.*

Now consider a (possibly empty) set $\mathcal{T}$ of formulas for which we have a transversal $\mathcal{M}$ of the $<^{\bar{P}}$–minimal models. We want to add new knowledge $\mathcal{U}$ to the theory $\mathcal{T}$ and thus obtain a transversal of the $<^{\bar{P}}$–minimal models of $\mathcal{T} \cup \mathcal{U}$. We call a function, which computes this new transversal, a *Revision Function*.

Definition 3.5 *Consider a first order language $\mathcal{L}$, where $I_{\mathcal{L}}$ is the set of all finite interpretations. Let $\mathcal{C}$ be a class of theories in $\mathcal{L}$. Let $f_{\bar{P}}$ be a function, which takes a theory, a set of models and a second theory as parameters and produces a set of models: $f_{\bar{P}} : \mathcal{C} \times 2^{I_{\mathcal{L}}} \times \mathcal{C} \rightarrow 2^{I_{\mathcal{L}}}$.*

$f_{\bar{P}}$ is called a Revision Function for *$\mathcal{C}$, iff for all $\mathcal{T}, \mathcal{U} \in \mathcal{C}$ and all transversals $\mathcal{M} \subseteq I_{\mathcal{L}}$ of $MOD^{\bar{P}}(\mathcal{T})/\sim^{\bar{P}}$:*

$f_{\bar{P}}(\mathcal{T}, \mathcal{M}, \mathcal{U})$ is a transversal of $MOD^{\bar{P}}(\mathcal{T} \cup \mathcal{U})/\sim^{\bar{P}}$.

In section 3.4 we will introduce an efficient revision function $Rev_{\bar{P}}$ and show its completeness for a large class of theories. The rest of this section is dedicated to the question how to use a revision function to comute minimal models and decide entailment under circumscription. Our first observation is that we can use the revision function directly to compute a transversal of the minimal models by executing the revision $Rev_{\bar{P}}(\emptyset, \{\emptyset\}, \mathcal{T})$. This method will be used in the PMON example (see section 3.5.1). In other applications we want to compute minimal models in multiple steps. That is, we already have a transversal $\mathcal{M}$ of the minimal models for a part $\mathcal{T}_0$ of the theory and we use $Rev_{\bar{P}}(\mathcal{T}_0, \mathcal{M}, \mathcal{T}_1)$ to compute the minimal models of $\mathcal{T}_0 \cup \mathcal{T}_1$. One reason for doing so can be the appearance of new knowledge. In this case we want to reuse the minimal models of the old theory to compute the new minimal models.

Another reason for computing minimal models in several steps is efficiency. In reasoning about action and change we can for example use standard model generation techniques to create a model of the static world, without regarding the action axioms. Then we revise this model with the action axioms. This technique is shown in section

3.5.2 using Baker's approach to non–monotonic reasoning. The most challenging application of our circumscription algorithm is the model–based diagnosis of technical systems. Solving diagnosis applications (see chapters 4 and 5), we have found that computing minimal models in several steps can focus and thereby speed up computation dramatically.

3.2.3 Deciding Entailment under Circumscription

Until now, we defined a method for computing the equivalence classes of the minimal models of a theory $\mathcal{T}$. These equivalence classes can be used to decide entailment under circumscription. The decision procedure is an application of theorem 3.2 by Lifschitz. Before we describe the general decision procedure, note that answering queries concerning predicates in $\bar{P}$ is now trivial, since a transversal of the $<^{\bar{P}}$–minimal models contains a model for each minimal combination of extensions of the predicates in $\bar{P}$.

Proposition 3.6 *Let $\bar{P} = (P_1, \ldots, P_n)$ a tuple of predicate constants, P_i ($i \in \{1, \ldots, n\}$) a predicate constant of arity r_i, and $\mathcal{M}$ a transversal of $MOD^{\bar{P}}(\mathcal{T})/\!\sim^{\bar{P}}$. An atom $P_i(x_1, \ldots, x_{r_i})$ follows from the circumscription of $\mathcal{T}$ in $\bar{P}$ varying all other predicates, iff $\forall M \in \mathcal{M} : M \models P_i(x_1, \ldots, x_{r_i})$.*

To prove that an arbitrary formula φ is entailed by the circumscription of $\mathcal{T}$ in $\bar{P}$, we show that $\neg\varphi$ does not hold in any $<^{\bar{P}}$–minimal model of $\mathcal{T}$. Our method for deciding entailment makes use of a *Filtering Function*. We can filter a set of equivalence classes with a formula φ by eliminating all equivalence classes, which do not contain a model of φ.

Definition 3.7 *Consider a language $\mathcal{L}$ and a class of theories $\mathcal{C}$ in $\mathcal{L}$. Let $\mathcal{T}$ be a theory, φ a formula and $\mathcal{M}$ a set of models.*

$$\Pi_{\bar{P}}(\mathcal{T}, \mathcal{M}, \varphi) := \{[M] | M \in \mathcal{M} \wedge \exists N \in [M] : N \models \varphi\}$$

A function $f_{\bar{P}} : \mathcal{C} \times 2^{I_{\mathcal{L}}} \times \mathcal{L} \to 2^{I_{\mathcal{L}}}$ is called a Filtering Function for *$\mathcal{C}$, iff for all $\mathcal{T} \in \mathcal{C}$, $\varphi \in \mathcal{L}$ (such that $\mathcal{T} \cup \{\varphi\} \in \mathcal{C}$) and for every transversal $\mathcal{M}$ of $MOD^{\bar{P}}(\mathcal{T})/\!\sim^{\bar{P}}$:*
$f_{\bar{P}}(\mathcal{T}, \mathcal{M}, \varphi)$ is a transversal of $\Pi_{\bar{P}}(\mathcal{T}, M, \varphi)$.

A filtering function $\mathit{Filter}_{\bar{P}}$ will be defined in section 3.4.6 as a simplified version of the revision function $\mathit{Rev}_{\bar{P}}$. Using the filtering function, we can decide whether an arbitrary formula φ follows from the circumscription by first computing a transversal of the minimal models and then filtering with $\neg\varphi$. If no equivalence class remains after the filtering, there is by definition no minimal model, in which $\neg\varphi$ holds. Thus, φ follows from the circumscription by theorem 3.2.

Theorem 3.8 *Let $f_{\bar{P}}$ be a filtering function for a class $\mathcal{C}$ of theories. Let $\mathcal{T} \in \mathcal{C}$ be a theory, $\mathcal{M}$ a transversal of $MOD^{\bar{P}}(\mathcal{T})/\sim^{\bar{P}}$ and φ a formula such that $\mathcal{T} \cup \{\varphi\} \in \mathcal{C}$.*

φ follows from the circumscription of $\mathcal{T}$ in $\bar{P}$ while varying all other predicates, iff $f_{\bar{P}}(\mathcal{T}, \mathcal{M}, \neg\varphi) = \emptyset$.

Proof:

"⇒" If φ follows from the circumscription, by theorem 3.2 φ holds in all $<^{\bar{P}}$–minimal models. We show that $\Pi_{\bar{P}}(\mathcal{T}, \mathcal{M}, \neg\varphi) = \emptyset$, which entails $f_{\bar{P}}(\mathcal{T}, \mathcal{M}, \neg\varphi) = \emptyset$.

Each $M \in \mathcal{M}$ is a $<^{\bar{P}}$–minimal model. Thus, every $N \in [M]$ is also a $<^{\bar{P}}$–minimal model. Since φ holds in all $<^{\bar{P}}$–minimal models, we conclude

$$\begin{array}{rl} & \forall M \in \mathcal{M} : \forall N \in [M] : N \models \varphi \\ \Rightarrow & \forall M \in \mathcal{M} : \nexists N \in [M] : N \models \neg\varphi \\ \Rightarrow & \{[M] | M \in \mathcal{M} \wedge \exists N \in [M] : N \models \neg\varphi\} = \emptyset \\ \Leftrightarrow & \Pi_{\bar{P}}(\mathcal{T}, \mathcal{M}, \neg\varphi) = \emptyset \end{array}$$

"⇐" $f_{\bar{P}}(\mathcal{T}, \mathcal{M}, \neg\varphi) = \emptyset$ is given. Since $f_{\bar{P}}(\mathcal{T}, \mathcal{M}, \neg\varphi)$ is a transversal of $\Pi_{\bar{P}}(\mathcal{T}, \mathcal{M}, \neg\varphi)$ by definition 3.7, we conclude that

$$\begin{array}{rl} & \Pi_{\bar{P}}(\mathcal{T}, \mathcal{M}, \neg\varphi) = \emptyset \\ \Rightarrow & \{[M] | M \in \mathcal{M} \wedge \exists N \in [M] : M \models \neg\varphi\} = \emptyset\} \\ \Rightarrow & \forall M \in \mathcal{M} : \forall N \in [M] : N \not\models \neg\varphi \end{array}$$

If we define $Models := \{M' | \exists M \in \mathcal{M} : M' \in [M]\}$, we can write this as

$$\forall M \in Models : M \not\models \neg\varphi$$

From the fact that $\mathcal{M}$ is a transversal of $MOD^{\bar{P}}(\mathcal{T})/\sim^{\bar{P}}$, we can conclude that *Models* contains all $<^{\bar{P}}$–minimal models, and thus $MOD^{\bar{P}}(\mathcal{T}) = Models$. Thus we have

$$\forall M \in MOD^{\bar{P}}(\mathcal{T}) : M \not\models \neg\varphi$$

We can now apply theorem 3.2 and conclude that φ follows from the circumscription. Q.E.D.

Some methods for reasoning about action and change [San94, Kar94] first minimize a certain predicate in a part of the theory (usually the domain axioms) and then filter the resulting models using another part of the theory (usually the observations). This can be formalized in our approach by applying the filtering operation twice. The first filtering uses the observations to prune the inappropriate models from the set of minimal models. To prove a query φ, a second filtering step with $\neg\varphi$ is used. See sections 3.5.1 and 3.5.2 for examples.

3.3 Variants of Circumscription

3.3.1 Keeping the extensions of certain predicates fixed

Up to now we have assumed that all predicates (except the minimized ones) vary during the minimization. *Fixed predicates* can be replaced by varying predicates using de Kleer's method [dK90a]: Instead of holding predicate Q fixed during the minimization of P, we define $Q'(\vec{x}) \equiv \neg Q(\vec{x})$, and then minimize P, Q and Q' in parallel.

3.3.2 Prioritized Circumscription

Prioritized Circumscription can be directly handled in our approach by multiple revisions. The semantical characterization of prioritized circumscription builds on the relation $\leq^{\bar{P}_1 > \ldots > \bar{P}_n}$ (compare [Lif86]).

Definition 3.9 *Let M_1, M_2 be models and $\bar{P}_1, \ldots, \bar{P}_n$ tuples of predicate constants. $M_1 \leq^{\bar{P}_1 > \ldots > \bar{P}_n} M_2$, iff*

1. *M_1 and M_2 have the same domain and agree in the interpretation of the constant symbols.*

2. *There is a $k \in \{0, \ldots, n\}$, such that*

   (a) $M_1 \sim^{\bar{P}_i} M_2$ for $i \in \{1, \ldots, k\}$ and

   (b) $M_1 <^{\bar{P}_k} M_2$

We write $M <^{\bar{P}_1 > \ldots > \bar{P}_n} N$, iff $M \leq^{\bar{P}_1 > \ldots > \bar{P}_n} N$ and not $N \leq^{\bar{P}_1 > \ldots > \bar{P}_n} M$.

A model M is called a $\leq^{\bar{P}_1 > \ldots > \bar{P}_n}$*–minimal model* of the theory $\mathcal{T}$, iff there is no model N of $\mathcal{T}$, such that $N <^{\bar{P}_1 > \ldots > \bar{P}_n} M$. We will denote the set of $<^{\bar{P}_1 > \ldots > \bar{P}_n}$–minimal models of $\mathcal{T}$ by $MOD^{\bar{P}_1 > \ldots > \bar{P}_n}(\mathcal{T})$. Lifschitz has shown in [Lif86] that $MOD^{\bar{P}_1 > \ldots > \bar{P}_n}(\mathcal{T})$ is exactly the set of models of the corresponding prioritized circumscription of $\mathcal{T}$

Theorem 3.10 *Let $\mathcal{T}$ be a theory and $\bar{P}_1, \ldots, \bar{P}_n$ set of predicate constants. $MOD^{\bar{P}_1 > \ldots > \bar{P}_n}(\mathcal{T})$ is the set of models of the prioritized circumscription of $\mathcal{T}$ in $\bar{P}_1 > \ldots > \bar{P}_n$, while varying all other predicates.*

From the definition of the logical entailment operator we can conclude, that a formula φ follows from the prioritized circumscription of $\mathcal{T}$ in $\bar{P}_1 > \ldots > \bar{P}_n$, iff φ holds in all models in $MOD^{\bar{P}_1 > \ldots > \bar{P}_n}(\mathcal{T})$. As in the case of the parallel circumscription we do not want to store all these models. Therefore, we compute only one model for any combination of minimal extensions of the predicates in $\bar{P}_1, \ldots, \bar{P}_n$.

Definition 3.11 *Let $\{\bar{P}_1,\ldots,\bar{P}_k\}$ be a set of tuples of predicate constants. For models M,N we define $M \sim^{\{\bar{P}_1,\ldots,\bar{P}_k\}} N$, iff $M \sim^{\bar{P}_i} N$ for all $i \in \{1,\ldots,k\}$.*

To answer queries concerning the prioritized circumscription we compute a transversal of the $<^{\bar{P}_1>\ldots>\bar{P}_n}$–minimal models with respect to the equivalence relation $\sim^{\bar{P}_1,\ldots,\bar{P}_n}$. Fortunately, we can compute this transversal by iterating a revision operator.

Definition 3.12 *$f_{\bar{P}_1>\ldots>\bar{P}_n}$.*
Let $\bar{P}_1,\ldots,\bar{P}_n$ be tuples of predicate constants, where $\bar{P}_i = (P_{i_1},\ldots P_{i_{m_i}})$. Let $\mathcal{T}$ be a theory and $\mathcal{U}$ a new information (also a theory). Let $\mathcal{T}' := \mathcal{T} \cup \mathcal{U}$. Let $\mathcal{M}_0$ be a transversal of $MOD^{\bar{P}_1}(\mathcal{T})$ and

$$\mathcal{M}_1 := f_{\bar{P}_1}(\mathcal{T},\mathcal{M}_0,\mathcal{U}) \tag{3.1}$$

For $i \in \{1,\ldots,n \Leftrightarrow 1\}$ we define

$$\mathcal{T}'^{i+1}_{Base} := \text{“All formulas in } \mathcal{T}' \text{ not containing predicates from } \bar{P}_{i+1}\text{”} \tag{3.2}$$

The new information $\mathcal{T}'^{i+1}_{New}(M)$ contains all formulas in $\mathcal{T}'$, in which predicates from $\bar{P}_{i+1}$ occur; furthermore it contains the extensions of the already minimized tuples of predicates ($\bar{P}_1,\ldots,\bar{P}_i$).

$$\begin{aligned}\mathcal{T}'^{i+1}_{New}(M) := \mathcal{T}' \backslash \mathcal{T}'^{i+1}_{Base} \cup \bigcup_{1\le j\le i} \bigcup_{1\le k\le m_i} & \{P_{j_k}(\vec{x}) | \vec{x} \in M|P_{j_k}|\} \\ \cup \quad & \{\neg P_{j_k}(\vec{x}) | \vec{x} \notin M|P_{j_k}|\}\end{aligned} \tag{3.3}$$

For a tuple of predicate constants $\bar{P} = (P_1,\ldots,P_m)$ we define $Del(M,\bar{P})$ as the model which agrees with M on all predicates except those in $\bar{P}$ and interprets all predicate constants in $\bar{P}$ as the empty predicate.

$$Del(M,\bar{P}) := M \backslash (\bigcup_{1\le k\le m} \{P_k(\vec{x}) | \vec{x} \in M|P_k|\}) \tag{3.4}$$

We now define $f_{\bar{P}_1>\ldots>\bar{P}_n}$ by a sequence of revisions:

$$\mathcal{M}_{i+1} := \bigcup_{M \in \mathcal{M}_i} f_{\bar{P}_{i+1}}(\mathcal{T}'^{i+1}_{Base}, \{Del(M,\bar{P}_{i+1})\}, \mathcal{T}'^{i+1}_{New}(M)) \tag{3.5}$$

$$f_{\bar{P}_1>\ldots>\bar{P}_n}(\mathcal{T},\mathcal{M}_0,\mathcal{U}) := M_n \tag{3.6}$$

We will now show that $f_{\bar{P}_1>\ldots>\bar{P}_n}$ returns a transversal of the $\le^{\bar{P}_1>\ldots>\bar{P}_n}$–minimal models. First, note that the preconditions for applying the revision operator are satisfied in the above definition.

Lemma 3.13 *In definition 3.12 the preconditions for applying the revision operator (defined in equation 3.5) are satisfied, i.e. if $M \in \mathcal{M}_i$ then $Del(M, \bar{P}_{i+1})$ is a transversal of $MOD^{\bar{P}_{i+1}}(\mathcal{T}'^{i+1}_{Base})/\sim^{\bar{P}_{i+1}}$.*

Proof: By induction on the number n of subsequent revisions we show that

1. For $n \geq 1$: $f_{\bar{P}_n}$ is applied to a theory $\mathcal{T}$ and a set $\mathcal{M}$ of models (and a new information φ), such that $\mathcal{M}$ is a transversal of $MOD^{\bar{P}_n}(\mathcal{T})$ and
2. every model $M \in \mathcal{M}_n$ is a model of $\mathcal{T}'$.

$n = 1$: In the equation 3.1 the preconditions for applying the revision function $f_{\bar{P}_1}$ are satisfied because we assume that $\mathcal{M}_0$ is a transversal of $MOD^{\bar{P}_1}(\mathcal{T})$. Thus, each $M \in \mathcal{M}_1$ is a model of $\mathcal{T}'$.

$n \to n+1$: We show that $\{Del(M, \bar{P}_{n+1})\}$ is a transversal of $MOD^{\bar{P}_{n+1}}(\mathcal{T}'^{i+1}_{Base})/\sim^{\bar{P}_{n+1}}$ for every $M \in \mathcal{M}_n$. By induction assumption, each $M \in \mathcal{M}_n$ is a model of $\mathcal{T}'$. M is also a model of $\mathcal{T}'^{n+1}_{Base}$ because $\mathcal{T}'^{n+1}_{Base} \subseteq \mathcal{T}'$ and $\mathcal{T}'$ is consistent. Since $\mathcal{T}'^{n+1}_{Base}$ does not contain predicates from $\bar{P}_{n+1}$, $Del(M, \bar{P}_{n+1})$ is also a model of $\mathcal{T}'^{n+1}_{Base}$. Furthermore it is a representative of the only $\leq^{\bar{P}_{n+1}}$–minimal model class (namely the one which interprets all predicates in $\bar{P}_{n+1}$ as the empty predicate). In summary, $\{Del(M, \bar{P}_{n+1})\}$ is a transversal of $MOD^{\bar{P}_{n+1}}(\mathcal{T}'^{i+1}_{Base})/\sim^{\bar{P}_{n+1}}$.

Next, we show that $\mathcal{M}_{n+1}$ contains only models of $\mathcal{T}'$: Following the definition of a revision function (3.5) each $M \in \mathcal{M}_{n+1}$ is a model of the theory together with the new information. For $n > 1$ this means that $\mathcal{M}_{n+1}$ contains only models of $\mathcal{T}'^{n+1}_{Base} \cup \mathcal{T}'^{n+1}_{New}(M')$ for $M' \in \mathcal{M}_n$. We have

$$\begin{aligned} \mathcal{T}'^{n+1}_{Base} \cup \mathcal{T}'^{n+1}_{New}(M') = \mathcal{T}' \cup \bigcup_{1 \leq j \leq n} \bigcup_{1 \leq k \leq m_j} & \{P_{j_k}(\vec{x}) \mid \vec{x} \in M'|P_{j_k}|\} \\ \cup & \{\neg P_{j_k}(\vec{x}) \mid \vec{x} \notin M'|P_{j_k}|\} \end{aligned} \tag{3.7}$$

The righthand side of this equation is satisfied by M', because M' is a model of $\mathcal{T}'$ (this follows from the induction assumption, because $M' \in \mathcal{M}_n$) and M' satisfies the conjunction of literals, which encodes only the extensions of certain predicates in M'. Thus, $\mathcal{M}_{n+1}$ contains only models of consistent supersets of $\mathcal{T}'$, which are in particular models of $\mathcal{T}'$.

Q.E.D.

Now we can prove the correctness of $f_{\bar{P}_1 > \ldots > \bar{P}_n}$.

Theorem 3.14 *Let $\mathcal{T}, \mathcal{U}$ be theories, $\bar{P}_1, \ldots, \bar{P}_n$ tuples of predicate constants, and $\mathcal{M}_0$ a transversal of $MOD^{\bar{P}_1}(\mathcal{T})$. Then, $f_{\bar{P}_1 > \ldots > \bar{P}_n}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$ is a transversal of $MOD^{\bar{P}_1 > \ldots > \bar{P}_n}(\mathcal{T} \cup \mathcal{U})/\sim^{\bar{P}_1, \ldots, \bar{P}_n}$*

Proof: We show by induction on the number i of predicate tuples: $f_{\bar{P}_1 > \ldots > \bar{P}_i}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$ is a transversal of $MOD^{\bar{P}_1 > \ldots > \bar{P}_i}(\mathcal{T} \cup \mathcal{U}) / \sim^{\bar{P}_1, \ldots, \bar{P}_i}$.

$i = 1$**:** Since M_0 is a transversal of $MOD^{\bar{P}_1}(\mathcal{T})$ and $f_{\bar{P}_1}$ is a revision function, definition 3.5 tells us that $f_{\bar{P}_1}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$ is a transversal of $MOD^{\bar{P}_1}(\mathcal{T} \cup \mathcal{U})$.

$i \rightarrow i+1$**:** a) Let us first show that $f_{\bar{P}_1, \ldots, \bar{P}_{i+1}}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$ contains a representative for every $\leq^{\bar{P}_1, \ldots, \bar{P}_{i+1}}$–minimal model M_{min} of $\mathcal{T} \cup \mathcal{U}$. Note, that M_{min} is also $\leq^{\bar{P}_1, \ldots, \bar{P}_i}$–minimal (see definition 3.9). By induction assumption, there is a representative of M_{min}'s $\sim^{\{\bar{P}_1, \ldots, \bar{P}_i\}}$–equivalence class in $\mathcal{M}_i$, i.e. there is a model $M^{\sim}_{min}$, such that $M_{min} \sim^{\{\bar{P}_1, \ldots, \bar{P}_i\}} M^{\sim}_{min}$.

Now we consider

$$\mathcal{M} := f_{\bar{P}_{i+1}}(\mathcal{T}'^{i+1}_{Base}, \{Del(M^{\sim}_{min}, \bar{P}_{i+1})\}, \mathcal{T}'^{i+1}_{New}(M^{\sim}_{min})). \tag{3.8}$$

Obviously, $\mathcal{M} \subset \mathcal{M}_{i+1}$, because $M^{\sim}_{min} \in \mathcal{M}_i$ (see the definition of $\mathcal{M}_{i+1}$ in equation 3.5). Because of lemma 3.13 we know that the conditions for applying $f_{\bar{P}_{i+1}}$ are satisfied in equation 3.8 and thus by definition 3.9 we conclude that $\mathcal{M}$ is a transversal of $MOD^{\bar{P}_{i+1}}(\mathcal{T}'^{i+1}_{New}(M^{\sim}_{min}) \cup \mathcal{T}'^{i+1}_{Base})$. The axioms in $\mathcal{T}'^{i+1}_{New}$ make sure that every model in $\mathcal{M}$ agrees with $M^{\sim}_{min}$ in the extensions of the predicates from $\bar{P}_1, \ldots, \bar{P}_n$. Furthermore, all models in $\mathcal{M}$ are models of $\mathcal{T} \cup \mathcal{U} = \mathcal{T}' \subseteq \mathcal{T}'^{i+1}_{New}(M^{\sim}_{min}) \cup \mathcal{T}'^{i+1}_{Base}$. In summary, $\mathcal{M}$ is a transversal of the $\bar{P}i+1$–minimal models out of all models which agree with $M^{\sim}_{min}$ on $\bar{P}_1, \ldots, \bar{P}_i$ and are models of $\mathcal{T}$. Since M_{min} is a $\leq^{\bar{P}_1 > \ldots > \bar{P}_i}$–minimal model of $\mathcal{T}$, it is represented in this transversal, thus there is an $M \in \mathcal{M}$, such that $M \sim^{\{\bar{P}_1, \ldots, \bar{P}_{i+1}\}} M_{min}$. This shows that every $\leq^{\bar{P}_1, \ldots, \bar{P}_{i+1}}$–minimal model is represented in $f_{\bar{P}_1 > \ldots > \bar{P}_{i+1}}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$.

b) Now we will show that $f_{\bar{P}_1, \ldots, \bar{P}_{i+1}}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$ contains only $\leq^{\bar{P}_1, \ldots, \bar{P}_{i+1}}$–minimal models of $\mathcal{T} \cup \mathcal{U}$. Our previous considerations under a) have shown that the models in equation 3.8 are $\leq^{\bar{P}_{i+1}}$–minimal within the models of $\mathcal{T} \cup \mathcal{U}$, which agree with $M^{\sim}_{min} \in \mathcal{M}_i$ in the extensions of the predicates from $\bar{P}_1, \ldots, \bar{P}_i$. Thus, they are $\leq^{\bar{P}_1, \ldots, \bar{P}_{i+1}}$–minimal. Since all models in $f_{\bar{P}_1 > \ldots > \bar{P}_{i+1}}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$ are computed as in equation 3.8, we conclude that $f_{\bar{P}_1 > \ldots > \bar{P}_{i+1}}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$ contains only $\leq^{\bar{P}_1, \ldots, \bar{P}_{i+1}}$–minimal models of $\mathcal{T} \cup \mathcal{U}$.

Together, a) and b) show that $f_{\bar{P}_1 > \ldots > \bar{P}_{i+1}}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$ is a set of $\leq^{\bar{P}_1, \ldots, \bar{P}_{i+1}}$–minimal models of $\mathcal{T} \cup \mathcal{U}$, which contains a representative of every class of $\leq^{\bar{P}_1, \ldots, \bar{P}_{i+1}}$–models of $\mathcal{T} \cup \mathcal{U}$. Thus, $f_{\bar{P}_1 > \ldots > \bar{P}_{i+1}}(\mathcal{T}, \mathcal{M}_0, \mathcal{U})$ is a transversal of $MOD^{\bar{P}_1 > \ldots > \bar{P}_n}(\mathcal{T} \cup \mathcal{U}) / \sim^{\bar{P}_1, \ldots, \bar{P}_n}$. Q.E.D.

As in the case of the parallel circumscription, we use the filtering operator for deciding queries concerning the prioritized circumscription. The extensions of all previously minimized predicates are considered part of the theory. We use the following notation:

Definition 3.15 $f_{\{\bar{P}_1,\ldots,\bar{P}_n\}}$
Let $f_{\bar{P}}$ be a filtering function. Let $\bar{P}_1 = (P_{11},\ldots,P_{1m_1})$, …, $\bar{P}_n = (P_{n1},\ldots,P_{nm_n})$ be tuples of predicate constants. Let

$$\bar{Q} := (P_{11},\ldots,P_{1m_1},\ldots,P_{n1},\ldots P_{nm_n})$$

Let $\mathcal{T}$ be a theory, $\mathcal{M}$ as set of models and φ a formula. We define

$$f_{\{\bar{P}_1,\ldots,\bar{P}_n\}}(\mathcal{T},\mathcal{M},\varphi) := f_{\bar{Q}}(\mathcal{T},\mathcal{M},\varphi)$$

We can decide, if a formula φ follows from the prioritized circumscription by filtering a transversal of the corresponding minimal models with $\neg\varphi$.

Theorem 3.16 *Let $\bar{P}_1,\ldots,\bar{P}_n$ be tuples of predicate constants. Let $f_{\bar{P}}$ be a filtering function for a class $\mathcal{C}$ of theories. Let $\mathcal{T} \in \mathcal{C}$ be a theory, $\mathcal{M}$ a transversal of $MOD^{\bar{P}_1>\ldots>\bar{P}_n}(\mathcal{T})/\sim^{\{\bar{P}_1,\ldots,\bar{P}_n\}}$, and φ a formula, such that $\mathcal{T} \cup \{\varphi\} \in \mathcal{C}$.*

φ follows from the circumscription of $\mathcal{T}$ in $\bar{P}_1 > \ldots > \bar{P}_n$, iff $f_{\{\bar{P}_1,\ldots,\bar{P}_n\}}(\mathcal{T},\mathcal{M},\neg\varphi) = \emptyset$.

Proof Sketch: The proof proceeds in analogy to the proof of theorem 3.8. $f_{\{\bar{P}_1,\ldots,\bar{P}_n\}}(\mathcal{T},\mathcal{M},\varphi) = \emptyset$ means that $\neg\varphi$ holds in no $\leq^{\bar{P}_1>\ldots>\bar{P}_n}$–minimal model. Thus, φ holds in all $\leq^{\bar{P}_1>\ldots>\bar{P}_n}$–minimal models and thus φ follows from the circumscription. Q.E.D.

3.4 Algorithms for Revision and Filtering

In the previous section we have reduced fixed predicates and prioritized circumscription to the base case of parallel circumscription of a theory in a tuple $\bar{P}$ of predicate constants while varying all predicates not in $\bar{P}$. We will now describe algorithms for the basic revision and filtering functions implementing this case of circumscription. These algorithms will work on a subset of first order logic, the fixed–domain theories. The material in this section was inspired by the work of Chou and Winslett on implementing model–based belief revision [CW94].

3.4.1 The Language

Two limitations are inherent in the model–based paradigm. (1) We can not handle infinite models. (2) We cannot handle infinitely many models. Both problems are avoided, if we limit ourselves to languages without function symbols and theories which include the unique name assumption and the domain closure assumption. Such theories are called *Fixed–Domain Theories*. In fixed–domain theories all models are isomorphic to Herbrand models [Luk90], thus it is sufficient to consider only the minimal Herbrand models in theorem 3.2, when using fixed–domain theories. Our algorithms can be used to handle two extensions of fixed–domain theories: functions symbols with Herbrand interpretation and infinite sorts. When using infinite sorts, we have to make sure that existential quantifiers only range over finite intervals, otherwise limitation (2) would be violated. The extensions are useful for reasoning with situation calculus and formalizing infinite integer time.

Every fixed domain theory can be represented in *Clause Form.*

Definition 3.17 *Clause Form. A* Clause *is a disjunction of literals $C = L_1 \vee \ldots \vee L_n$. Every variable in C is implicitly quantified.*

A theory $\mathcal{S}$ is in Clause Form*, iff it is represented by a set of clauses $\mathcal{S} = \{C_1, \ldots, C_m\}$.*

However, the usual transformation of a first order formula into clause form [Luk90] is not useful, because it introduces function symbols (skolem functions) into the theory during the elimination of existential quantifiers. Instead of using this transformation, we first transform each formula into *Prenex Conjunctive Normal Form* [Luk90]. Then, we eliminate the existential quantifiers through instantiation.

Definition 3.18 *Prenex Conjunctive Normal Form [Luk90]. A formula F is in* Prenex Conjunctive Normal Form*, iff*

$$F = Q_1 x_1 \ldots Q_n x_n \left(\bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} L_{ij} \right), \textit{ where}$$

$Q_i \in \{\exists, \forall\}$ for all $i \in \{1, \ldots, n\}$, $x_1, \ldots, x_n$ are distinct variables occurring in $\bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} L_{ij}$, and L_{ij} are literals for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, k_i\}$.

Algorithm 3.19 *Conversion of a Fixed Domain Theory into Clause Form. Let $\mathcal{T} = \{F_1, \ldots, F_n\}$ be a fixed domain theory. Let $\{c_1, \ldots, c_t\}$ be the set of all constants occurring in $\mathcal{T}$. For every formula $F \in \mathcal{T}$ do the following:*

1. *Transform F into prenex conjunctive normal form. Call the resulting formula F_1.*

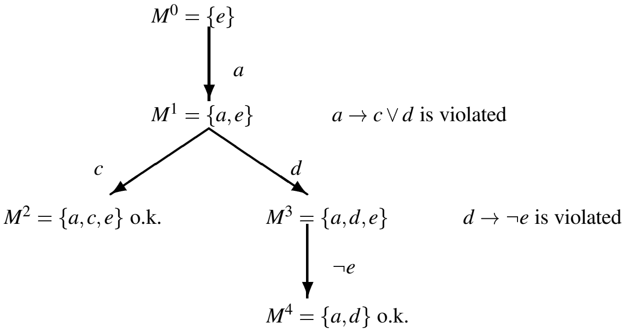
2. *Instantiate every existential quantifier in F_1 by replacing $\exists x \varphi$ with $\bigvee_{c \in \{c_1, \ldots, c_t\}} \varphi_{[x/c]}$. Call the resulting formula F_2.*

3. *Distribute the universal quantifiers in F_2 over the conjunctions and denote the result by a set of clauses.*

From now on, we assume that all fixed domain theories used as input to our algorithms are given in clause form.

3.4.2 Repairing Inconsistent Models

Consider a given model M^0 of a theory $\mathcal{S}$. We want to augment $\mathcal{S}$ by the new information (set of clauses) $\mathcal{U}$ giving an extended theory $\mathcal{S}' = \mathcal{S} \cup \mathcal{U}$. Consider the theory $\mathcal{S} = \{a \rightarrow c \vee d, d \rightarrow \neg e\}$, in clause form $\{\{\neg a, c, d\}, \{\neg d, \neg e\}\}$ together with the model $M^0 = \{e\}$ and the new information $\mathcal{U} = \{\{a\}\}$.



Since M^0 does not contain the new unit clause a, we insert a into the model, giving a new model M^1. M^1 contradicts the axiom $a \rightarrow c \vee d$. We can change it in two ways in order to make it satisfy the axiom again: Either we assume c or d, because these are the other literals in the corresponding clause, which can make it true again. While assuming c creates a consistent model (M^2) of $\mathcal{S}'$, assuming d violates the axiom $d \rightarrow \neg e$. Thus, we have to delete e in order to obtain the consistent model M^4.

To formalize the iterative model repair algorithm motivated in the above example, we need the concept of *Committed Literals*. The positive and negative ground literals, which have been inverted in model M^i compared to the initial model M^0 are called committed literals $Comm(M^i)$.

Definition 3.20
$Comm(M^i) := \{l | l \textit{ is a ground literal and } M^i \models l \textit{ and } M^0 \not\models l\}$

In our example $Comm(M^3) = \{a, d\}$. To avoid cycles in the model repair algorithm, a committed literal is not considered for inversion again. We consider a literal for inversion, if it occurs in a violated clause and neither the literal itself nor its negative counterpart are committed. Such a literal is called a *Flipping*.

Definition 3.21 *Let M^i be a model and l a ground literal. l is called a* Flipping, *iff*

1. *There exists a clause C in $\mathcal{S}$ and a substitution σ such that $l \in C\sigma$ and $M^i \not\models C\sigma$ and*

2. *$\neg l \notin Comm(M^i)$.*

Obviously, if there is no flipping for a model M^i, it cannot be repaired without changing already committed literals, and thus the current branch in the search tree of the repair algorithm contains no model of $\mathcal{S}'$. We can specify a stronger criterion for detecting that M^i can no longer be repaired.

Lemma 3.22 *If there exist a clause $C \in \mathcal{S}'$ and a substitution σ, such that $M^i \not\models C\sigma$ and no literal $l \in C\sigma$ is a flipping then there is no model M of $\mathcal{S}'$, such that $Comm(M) \supseteq Comm(M^i)$.*

Proof: If $M^i \not\models C\sigma$, and no literal $l \in C\sigma$ is a flipping, then for each literal $l \in C\sigma$, $\neg l$ must be in $Comm(M^i)$. Now, consider a model M, such that $Comm(M) \supseteq Comm(M^i)$. Then, trivially, for each literal $l \in C\sigma$, $\neg l$ is also in $Comm(M)$. Thus, $M \not\models C\sigma$ (because the negations of all literals in $C\sigma$ are true in M) and consequently $M \not\models \mathcal{S}'$, because C is in $\mathcal{S}'$. Q.E.D.

In each repair step, our algorithm considers all possible flippings. A branch in the algorithm's search tree is terminated if either a consistent model is found, or the branch provably contains no solution.

Definition 3.23 *Let $\mathcal{S}$ be a theory and M an interpretation.*

1. *If $M \models \mathcal{S}$: $Step(\mathcal{S}, M) := \emptyset$.*
2. *If $M \not\models \mathcal{S}$:*
   (a) *If there exists a clause $C \in \mathcal{S}$ and a substitution σ such that $M \not\models C\sigma$ and no literal within $C\sigma$ is a flipping: $Step(\mathcal{S}, M) := \{\emptyset\}$.*

   (b) *Otherwise,*

   $$Step(\mathcal{S}, M) := \{M' | M' \text{ is obtained from } M \text{ by inverting the truth value of a flipping } l\}$$

Case 2(a) corresponds to an inconsistency, which can no longer be repaired, because all literals in the violated clause are already committed.

3.4.3 Revision Algorithm

Now we are ready to present an algorithm $Rev_{\bar{P}}$, which is a revision function for fixed–domain theories. This algorithm applies a sequence of repair steps to every model $M \in \mathcal{M}$ and discards the non-minimal models. By executing the repair steps in a best–first order it limits the generation of non–minimal models.

To repair a model M the algorithm $Rev_{\bar{P}}$ iteratively applies the *Step*–function. It maintains a set *WM* of inconsistent models and a set *Solutions* of consistent models. Initially $WM = \{M\}$ and $Solutions = \emptyset$. The algorithm now selects a $<^{\bar{P}}$–minimal inconsistent model $\hat{M}$ from *WM*, i.e. $\hat{M}$ has the property that there is no $M' \in WM \cup Solutions$ so that $M' <^{\bar{P}} \hat{M}$. Then, $\hat{M}$ is deleted from *WM* and $Step(\mathcal{S}, \hat{M})$ is computed.

If $Step(\mathcal{S}, \hat{M}) = \emptyset$ then $\hat{M}$ is consistent, i.e. $\hat{M} \models \mathcal{S}$. The model $\hat{M}$ is inserted into the solutions set and can be used to delete non–minimal models from *WM*: All models, in which the extensions of the predicates in $\bar{P}$ are equivalent to or larger than in $\hat{M}$ can be deleted, because we want to obtain only one solution out of each minimal

$\sim^{\bar{P}}$-class. If $\mathcal{N} := Step(\mathcal{S},\hat{M}) \neq \emptyset$, then the models from $\mathcal{N}$ are inserted into WM. If $Step(\mathcal{S},\hat{M}) = \{\emptyset\}$, no further repair step can be applied to the inconsistent model $\hat{M}$ and it is discarded. The revision of M is finished, if $WM = \emptyset$. This algorithm (*RepairModel*) has to be applied to every model in the set of initial models $\mathcal{M}$. Finally, another $<^{\bar{P}}$–minimality check is performed on the result models from the individual revisions.

Algorithm 3.24 $Rev_{\bar{P}}(\mathcal{S}, M, \mathcal{U})$

```
FUNCTION $Rev_{\bar{P}}(\mathcal{S}, \mathcal{M}, \mathcal{U})$
$Solutions := \emptyset$;
$\mathcal{S}' := \mathcal{S} \cup \mathcal{U}$;
FOR EACH $M \in \mathcal{M}$ DO
    $Solutions := Solutions \cup RepairModel(\mathcal{S}', M)$;
FOR EACH $S \in Solutions$ DO
    IF $\exists S' \in Solutions \backslash S : S' <^{\bar{P}} S$ THEN
        $Solutions := Solutions \backslash S$;
RETURN $Solutions$;

FUNCTION $RepairModel(\mathcal{S}, M)$
$WM := \{M\}$;  $Solutions := \emptyset$;
WHILE $WM \neq \emptyset$ DO
    Select a $<^{\bar{P}}$–minimal model $\hat{M}$ from $WM$;
    $WM := WM \backslash \{\hat{M}\}$;
    $\mathcal{N} := Step(\mathcal{S}, \hat{M})$;
    IF $\mathcal{N} = \emptyset$ THEN
        IF $\nexists M \in Solutions : M <^{\bar{P}} \hat{M}$ THEN
            $Solutions := Solutions \cup \hat{M}$;
    (*)     $WM := WM \backslash \{M \in WM : \hat{M} <^{\bar{P}} M \vee \hat{M} \sim^{\bar{P}} M\}$;
    ELSE IF $\mathcal{N} \neq \{\emptyset\}$ THEN $WM := WM \cup \mathcal{N}$;
    ELSE Discard $\hat{M}$
RETURN $Solutions$;
```

3.4.4 Properties of the Algorithm

In this section we will develop soundness and completeness results for the revision function $Rev_{\bar{P}}$. Our goal is to show that $Rev_{\bar{P}}$ is a revision function for fixed domain theories. We first have to prove the following properties:

Termination: The algorithm always terminates for fixed–domain theories.

Correctness: All models returned by the algorithm are minimal models of the theory together with the new literal.

Completeness: The algorithm outputs one model from each $\sim^{\bar{P}}$–equivalence class.

For fixed–domain theories we can show termination of algorithm 3.24 by proving that *Step* can only derive a finite number of successor models from a given model M.

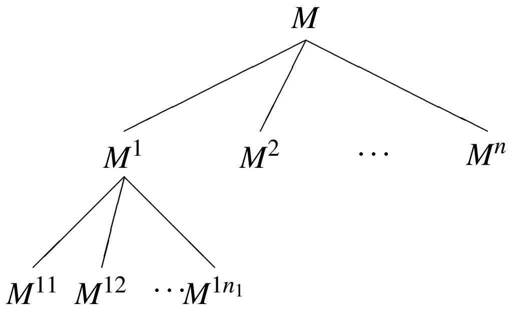
Proposition 3.25 *Algorithm 3.24 terminates for fixed domain theories.*

Proof: Since the given transversal $\mathcal{M}$ contains only finitely many models, *RepairModel* is only called finitely often and it is sufficient to show that *RepairModel* terminates for every fixed domain theory. *RepairModel* uses the set *WM*, which initially only contains one model M. Every model occurring in *WM* is obtained by iterative application of *Step*, starting with M. We show that only finitely many models can be generated in this fashion. We define a graph $G = \langle V, E \rangle$ as follows:

V**:** is the set of all models derivable by iterative application of *Step* to M.

E**:** $(M^i, M^j) \in E$, iff $M^j \in Step(\mathcal{S}, M^i)$.

First we show that this graph is really a tree as depicted in the following figure:



A tree is a connected graph without cycles. Obviously, G is connected, because we explicitly consider only those models as nodes, which can be derived using *Step*. We prove by contradiction that there is no cycle in G: Consider a cycle $M^i, M^{i+1}, \ldots, M^{i+r}, M^i$ in G. We know that *Step* always adds new committed literals. Thus, $Comm(M^j) \subset Comm(M^{j+1})$ for every model in the cycle. Transitivity of $\subset$ tells us that $M^i \subset M^{i+r}$. Since we assumed a cycle we also conclude $M^{i+r} \subset M^i$, which is a contradiction.

Leaves of the tree G are consistent nodes ($Step(M^x) = \emptyset$) or nodes which cannot be repaired because all literals, which could be changed are already committed ($Step(M^x) = \{\emptyset\}$). We show that this tree is finite by proving that every branch is finite and every node has finite arity.

1. Every application of *Step* commits the truth value of some literals, consequently the set of of committed literals grows monotonically ($N \in Step(\mathcal{S}, M) \Rightarrow Comm(N) \supset Comm(M)$). Since the number of ground literals is finite in a fixed–domain theory, every branch must finally lead to a consistent node, or a node which cannot be repaired. Thus, every branch is finite.

2. The number of ground literals (and thus flippings) is finite in a fixed domain theory. Thus, every node has finite arity.

The number of models generated by *Step* is finite. The models at the end of each finite branch are either stored as solutions or discarded. So, the termination condition of the `WHILE`–loop becomes true after finite time and *RepairModel* terminates. Q.E.D.

Our next result shows that all models generated by the algorithm are minimal models of $\mathcal{S}$. This result follows easily from the definitions.

Proposition 3.26 *Let $\mathcal{S}, \mathcal{U}$ be fixed–domain theories and $\mathcal{M}$ a transversal of $MOD^{\bar{P}}(\mathcal{S})/\sim^{\bar{P}}$. Then $Rev_{\bar{P}}(\mathcal{S}, \mathcal{M}, \mathcal{U}) \subseteq MOD^{\bar{P}}(\mathcal{S} \cup \mathcal{U})$.*

Proof: Every model M returned by *RepairModel* is a model of $\mathcal{S} \cup \mathcal{U}$, as for each such model $Step(\mathcal{S} \cup \mathcal{U}, M) = \emptyset$. The non–minimal models possibly generated by *RepairModel* are discarded in the loop at the end of $Rev_{\bar{P}}(\mathcal{S}, \mathcal{M}, \mathcal{U})$. Q.E.D.

To simplify the following completeness proof, let us first show that the pruning step (marked by (*) in algorithm 3.24) never entirely deletes a minimal equivalence class.

Lemma 3.27 *The pruning steps are correct, i.e. no $\sim^{\bar{P}}$–equivalence class of a $<^{\bar{P}}$–minimal model of $\mathcal{S} \cup \mathcal{U}$ is deleted entirely.*

Proof: If M is pruned, then there exists $M' \in WM$, such that either (1) $M' <^{\bar{P}} M$ or (2) $M' \sim^{\bar{P}} M$. In case (1), M is not a minimal model of $\mathcal{S} \cup \mathcal{U}$, in case (2) we have $[M] = [M']$, thus M' is still representing the equivalence class of M, and M can be deleted. Q.E.D.

Having shown the correctness of the pruning steps, we will not consider them in the following completeness proof. For the completeness of the algorithm it is crucial that the repair steps are exhaustive in the sense that the branches generated by *Step* account for any every consistent model of $\mathcal{S} \cup \mathcal{U}$. This is formalized in the next lemma.

Lemma 3.28 *(Case Splitting Property) Let M be a model of $\mathcal{S}$ and M' an interpretation which is no model of $\mathcal{S}$. If $M \models Comm(M')$ then there exists an $N \in Step(\mathcal{S}, M')$, such that $M \models Comm(N)$.*

Proof: We assumed $M' \not\models \mathcal{S}$: Thus, there is a clause $C \in \mathcal{S}$ and a substitution σ, such that $C\sigma$ is ground and $M' \not\models C\sigma$ (otherwise M' would be a model). Since M is a model of $\mathcal{S}$ it has to differ from M' in at least one ground literal occuring in $C\sigma$. This literal l is not committed in M' since M and M' agree on all committed literals of M', consequently l is a flipping and the model created by flipping l in M' is in $Step(\mathcal{S}, M')$. Q.E.D.

Using the case splitting property, we can now show the completeness of our algorithm for fixed–domain theories.

Proposition 3.29 *Let $\mathcal{S},\mathcal{U}$ be fixed–domain theories and $\mathcal{M}$ a transversal of $MOD^{\bar{P}}(\mathcal{S})/\sim^{\bar{P}}$. Then for every $M \in MOD^{\bar{P}}(\mathcal{S})$ there exists $N \in Rev_{\bar{P}}(\mathcal{S},\mathcal{M},\mathcal{U})$, such that $M \sim^{\bar{P}} N$.*

Proof: Since M is a $<^{\bar{P}}$–minimal model of $\mathcal{S} \cup \mathcal{U}$, there exists a $<^{\bar{P}}$–minimal model N of $\mathcal{S}$, such that $N \leq^{\bar{P}} M$ (Note, that M is in particular a model of $\mathcal{S}$. Either there is a model with a smaller extension of the predicates in $\bar{P}$ than M or M itself is a minimal model of $\mathcal{S}$). Since $\mathcal{M}$ is a transversal of $MOD^{\bar{P}}(\mathcal{S})/\sim^{\bar{P}}$ there is a model N' in $\mathcal{M}$, such that $N' \sim^{\bar{P}} N$. Consider the revision tree of such an N'. In the beginning $Comm(N') = \emptyset$ thus $M \models Comm(N')$. Because of the case splitting property there is a path $N^{(0)} := N', N^{(1)}, N^{(2)}, \ldots$ such that

1. $N^{(i+1)} \in Step(\mathcal{S}, N^{(i)})$.
2. $\forall i : M \models Comm(N^{(i)})$.

On this path the set of committed literals grows. In every step some literals from M are added. So we know that there is a consistent model on this path (because M is consistent and will finally occur on the path). We also know that there is no consistent model of $\mathcal{S} \cup \{\varphi\}$ with smaller extensions of the predicates in $\bar{P}$ than M. We can conclude that there finally is a consistent a model $N^{(r)}$ on the path with $N^{(r)} \sim^{\bar{P}} M$. Q.E.D.

Propositions 3.26 and 3.29 together with the observation that at most one representative is computed for every equivalence class show that $Rev_{\bar{P}}$ computes a transversal of the minimal models and consequently is a revision function for fixed domain theories.

Theorem 3.30 *$Rev_{\bar{P}}$ is a revision function for fixed–domain theories.*

Algorithm 3.24 also computes minimal Herbrand models in the presence of functions symbols and infinite sorts. When we convert it from a best first to a breadth first algorithm by changing the management of the node list *WM* we can show that it always finds a set of finite minimal Herbrand models, if it exists. There are two directions for further research: (1) extending the implementation by function symbols with more general interpretation using the ideas from [CW94], (2) to further investigate the connection between Herbrand models of theories with infinite sorts to circumscription. The correspondence is less obvious then in the case of fixed–domain theories, because DCA and UNA are not expressible as first order sentences in such theories.

3.4.5 An Iterative Deepening Algorithm

The function *RepairModel* in algorithm 3.24 implements a best first search for minimal models. Although this is a very natural solution, it has some efficiency drawbacks.

- Models have to be compared using a time–consuming subset–check on the extensions of the predicates in $\bar{P}$ to find a $<_{\bar{P}}$–minimal model.
- The list of models has bad locality. Models have to be selected/inserted at any place in the list,

Both disadvantages are avoided without affecting the semantics by using the following iterative deepening algorithm. With each model M we associate a value $c(M)$ which is the number of atoms for the predicates in $\bar{P}$ contained in M:

Definition 3.31 *$c(M)$*
Let $\bar{P} = (P_1, \dots, P_n)$ be a tuple of predicate constants.

$$c(M) = \sum_{1 \leq i \leq n} |M|P_i||,$$

where $|M|P_i||$ denotes the number of elements in the extension of P_i in M.

For this algorithm the model set is organized as a stack with the usual operations Push and Pop. A stack has a better locality than the list of models used in the best–first algorithm, because all operations only affect its first element. When using a stack, we can implement the push and pop operations without really copying the models. However, since its top element is just some arbitrary model, we must implement other mechanisms to avoid the generation of non–minimal models. The iterative deepening algorithm first defines an initial upper bound *Cutoff* $= 0$ on $c(M)$. If a partially repaired model M has $c(M) >$ *Cutoff* it is deleted, but a flag is set to indicate the deletion. The algorithm computes all consistent models with $c(M) = 1$ and then sets *Cutoff* to the smallest $c(M)$ of a model M which was deleted. The algorithm proceeds until no model is deleted because of the cutoff–value. Algorithm 3.32 summarizes the procedure. For some applications (like diagnosis) it is useful to restrict $c(M)$ by some absolute bound (see chapter 2) and discard all models M completely which have a greater $c(M)$.

Algorithm 3.32 *ID_Repair_Model*

```
FUNCTION $ID\_Repair\_Model$($\mathcal{S}$, M)
$Solutions$ := $\emptyset$;
$Cutoff$ := 0;
$Next\_Cutoff$ := 0;
WHILE $Next\_Cutoff > Cutoff$ DO
    $Models$ := $\emptyset$;
    $Models.Push(M)$;
    $Cutoff$ := $Next\_Cutoff$;
    WHILE $Models \neq \emptyset$ DO
        $Models.Pop(\hat{M})$;
        $\mathcal{N}$ := $Step(\mathcal{S},\hat{M})$;
```

```
            IF $\mathcal{N} = \emptyset$ THEN
               IF $\nexists M \in Solutions : M \leq^{\bar{P}} \hat{M}$ THEN
                  $Solutions := Solutions \cup \{\hat{M}\}$;
               END IF;
            ELSE IF $\mathcal{N} \neq \{\emptyset\}$ THEN
               FOR ALL $M \in \mathcal{N}$ DO
                  IF $c(M) > Cutoff$ then
                     $Discard\, M$;
                     $Set\ Next\_Cutoff\ to\ the\ smallest$
                     $value\ c(M)\ in\ a\ discarded\ model\ so\ far$;
                  ELSE $Models.Push(M)$;
                  END IF;
               END FOR;
            END IF;
         END WHILE;
      END WHILE;
      RETURN $Solutions$;
```

3.4.6 Filtering Algorithm

For the filtering we use the same *Step*–repair function as for the revision. Instead of minimizing the extensions of the predicates in $\bar{P}$, we now hold the previously minimized extensions of the predicates in $\bar{P}$ fixed, by encoding them as part of the theory.

The filtering algorithm tries to find one model for every equivalence class in $\mathcal{M}$. Equivalence classes are filtered out, when no consistent model is found. In the algorithm this corresponds to the last ELSE–statement, which denotes that a model is discarded if it is inconsistent and there are no further repair steps applicable, because in at least one inconsistency all literals are committed.

Algorithm 3.33 $Filter_{\bar{P}}(\mathcal{S}, M, \varphi)$

```
FUNCTION $Filter_{\bar{P}}(\mathcal{S}, \mathcal{M}, \varphi)$
$Solutions := \emptyset$;
FOR EACH $M \in \mathcal{M}$ DO
    $\mathcal{S}' := \mathcal{S} \cup \{\varphi\} \cup \bigcup_{1 \leq i \leq n} \left( \bigcup_{\vec{x}:M \models P_i(\vec{x})} P_i(\vec{x}) \cup \bigcup_{\vec{x}:M \not\models P_i(\vec{x})} \neg P_i(\vec{x}) \right)$;
    $Solutions := Solutions \cup RepairModel'(\mathcal{S}', M)$;
RETURN $Solutions$;

FUNCTION $RepairModel'(\mathcal{S}', M)$
$WM := \{M\}$;  $S := \emptyset$;
WHILE $WM \neq \emptyset$ DO
    Select a model $\hat{M}$ from $WM$;
    $WM := WM \setminus \{\hat{M}\}$;
    $\mathcal{N} := Step(\mathcal{S}' \cup \{L\}, \hat{M})$;
    IF $\mathcal{N} = \emptyset$ THEN $S := \{\hat{M}\}$; $WM := \emptyset$;
    ELSE IF $\mathcal{N} \neq \{\emptyset\}$ THEN $WM := WM \cup \mathcal{N}$;
    ELSE Discard $\hat{M}$;
RETURN $S$;
```

As for the revision algorithm we have to show termination, correctness and completeness. We can reuse some of the proofs for the revision algorithm in order to show these properties.

Proposition 3.34 *Algorithm 3.33 terminates for every fixed domain theory $\mathcal{S}$.*

Proof: From Proposition 3.25 we know that the set of models derivable from the initial model $\hat{M}$ is finite. Since the number of committed literals in these models grows monotonically, iterative application of *Step* will lead to models which either contain an inconsistency which consists only of committed literals (last `ELSE`–case) or to a consistent model (first `IF`–case). Q.E.D.

Before we formulate the completeness result for the filtering algorithm, note that filtering does not include a minimization, i.e. we do not want to compute the minimal models of $\mathcal{S} \cup \{\varphi\}$ but rather check, if a minimal model of $\mathcal{S}$ exists, which is a model of φ.

Proposition 3.35 *Let $\mathcal{S}$ be a fixed–domain theory, φ a formula such that $\mathcal{S} \cup \{\varphi\}$ is a fixed-domain theory, and $\mathcal{M}$ a transversal of $MOD^{\bar{P}}(\mathcal{S})/\sim^{\bar{P}}$. Then $Filter_{\bar{P}}(\mathcal{S}, \mathcal{M}, \varphi) \subseteq MOD^{\bar{P}}(\mathcal{S})$.*

Proof: Every model M returned by *RepairModel′* satisfies $Step(\mathcal{S} \cup \{\varphi\}, M) = \emptyset$. Thus, it is a model of $\mathcal{S} \cup \{\varphi\}$ and in particular a model of $\mathcal{S}$. M is minimal, because it has the same extension of the predicates in $\bar{P}$ as the minimal model, from which it was derived. Q.E.D.

Proposition 3.36 *Let $\mathcal{S}$ be a fixed–domain theory, φ a formula such that $\mathcal{S} \cup \{\varphi\}$ is a fixed-domain theory, and $\mathcal{M}$ a transversal of $MOD^{\bar{P}}(\mathcal{S})/\sim^{\bar{P}}$. Then for every $M \in MOD^{\bar{P}}(\mathcal{S})$, such that $M \models \varphi$, there exists $N \in Filter_{\bar{P}}(\mathcal{S}, \mathcal{M}, \varphi)$, such that $M \sim^{\bar{P}} N$.*

Proof: Since M is a $<^{\bar{P}}$–minimal model of $\mathcal{S}$, there exists a model $N \in \mathcal{M}$ such that $N \sim^{\bar{P}} M$. Consider the revision tree of such an N'. Analogous to the proof of proposition 3.29 we can use the case splitting property to show that there is a path leading finally to M on which we encounter M or another consistent model appearing earlier on the path. Since we fixed the extension of the predicates in $\bar{P}$ during filtering, we know that the model $N^{(r)}$ we find will be in the same $\sim^{\bar{P}}$–equivalence class as M. Q.E.D.

Let us summarize: From proposition 3.35 we know that all models returned by $Filter_{\bar{P}}$ are minimal models of $\mathcal{S}$, which make φ true. Moreover, proposition 3.36 tells us that $Filter_{\bar{P}}$ find a model for each $\sim^{\bar{P}}$–equivalence class. Thus, $Filter_{\bar{P}}$ computes a transversal of the remaining equivalence classes and we conclude:

Theorem 3.37 *$Filter_{\bar{P}}$ is a filtering function for fixed–domain theories.*

In section 3.3 we have reduced circumscription with fixed predicates and prioritized circumscription to parallel circumscription with varying predicates. In this section we have defined algorithms for parallel circumscription with varying predicates. Together, both steps provide a powerful computation framework for many variants of circumscription. Before we move on to the formalization of diagnostic concepts within this framework, we study proof–of–concept applications from the non–monotonic reasoning domain.

3.5 Non–monotonic Reasoning Applications

To demonstrate the flexibility of DRUM–II, we apply it to some problems from the "reasoning about action and change" domain. Reasoning about action and change is a research area within the temporal reasoning field, where non–monotonic reasoning is used to infer intuitively correct conclusions from logical axiom formalizing the effects of actions. Many formalisms have been proposed in this area. We implement both Sandewall's and Baker's approaches to reasoning about action and change and Kartha's [Kar94] extension to Baker's approach.

Furthermore, we solve two formalizations of "Nixon's diamond" [Gin89, WS97], a well–known non–monotonic reasoning problem. The efficiency of our implementation is discussed in section 3.5.5.

3.5.1 PMON–Circumscription

In this section we show how to implement Sandewall's approach to reasoning about action and change [San94]. We use PMON–circumscription (recently discussed by Doherty [Doh94]). This example underlines the importance of reasoning with equivalence classes and the efficient use of fixed predicates. We will use the Russian Turkey Shoot Scenario: There are two fluents, a (alive) and l (loaded) and three actions, *Load*, *Spin* (spinning the guns chamber) and *Fire*. The gun was loaded between the time points 1 and 2, the spinning action was performed between 3 and 4 and the gun was fired between 5 and 6.

In Sandewall's framework the effect of actions is described by *Reassignment Formulas*, e.g. the loading action is described by $[1;2]l := T$ which means that the fluent l is assigned the value true during [1;2]. Reassignment formulas can include a condition. The firing action is described by the formula $[5]l \to [5,6](\neg a := T \wedge \neg l := T)$ denoting that a and l are false after the shooting if the gun was loaded at time 5. Before reasoning, the reassignment formulas are transformed into first order logic. Two predicates are used: $Holds(t,f)$, (fluent f holds at time t) and $Occlude(t,f)$ (fluent f may change its value at time point t as effect of an action). PMON–Circumscription first minimizes *Occlude* while holding *Holds* fixed. Then the resulting models are filtered with the observations and the nochange axiom, which states that a fluent f can only change its value at time t, if $Occlude(t,f)$ holds[1].

$$NCP : \forall f,t\ Holds(t,f) \oplus Holds(t+1,f) \rightarrow Occlude(t+1,f)$$

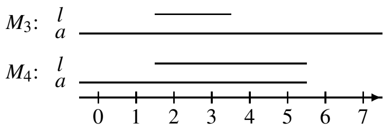
This reasoning pattern can be directly simulated in our approach. However, fixing the whole extension of *Holds* leads to a large amount of incomparable models. We extend the transformation to first order logic by generating a literal $Precond(I)$ for each conditional reassignment formula. The truth value of $Precond(I)$ is coupled with the truth value of the condition in the reassignment formula. In our example we generate such a literal for the *Fire*–action: $Precond(1) \equiv Holds(5,l)$. We obtain the correct results by holding *Precond* fixed instead of *Holds*. The translation yields theory *SCD*:

$$\begin{aligned}
&\exists t(1 \leq t < 2) \wedge \forall t'(t < t' \leq 2 \rightarrow Holds(t',l))\\
&\qquad \wedge \forall t'(1 < t' \leq 2 \rightarrow Occlude(t',l))\\
&\exists t(3 \leq t < 4) \wedge \forall t'(t < t' \leq 4 \rightarrow (Holds(t',l) \vee \neg Holds(t',l)))\\
&\qquad \wedge \forall t'(3 < t' \leq 4 \rightarrow Occlude(t',l))\\
&Holds(5,l) \rightarrow ((\exists t.5 \leq t < 6 \wedge \forall t'(t \leq t' < 6 \rightarrow \neg Holds(t',a)))\\
&\qquad\qquad\qquad \wedge \forall t'(5 < t' \leq 6 \rightarrow Occlude(t',a)) \wedge\\
&\qquad\qquad\qquad (\exists t.5 \leq t < 6 \wedge \forall t'(t \leq t' < 6 \rightarrow \neg Holds(t',l)))\\
&\qquad\qquad\qquad \wedge \forall t'(5 < t' \leq 6 \rightarrow Occlude(t',l)))\\
&Precond(1) \equiv Holds(5,l)\\
&Precond(1) \equiv \neg Precond'(1)
\end{aligned}$$

We first compute $Rev_{(Occlude,Precond,Precond')}(\emptyset,\{\emptyset\},SCD)$ and obtain two result models:

$$\begin{aligned}
M_1 :\ & \{Holds(2,l), Occlude(2,l), Occlude(4,l), Precond'(1)\}\\
M_2 :\ & \{Holds(2,l), Holds(5,l), Occlude(2,l), Occlude(4,l),\\
& Occlude(6,a), Occlude(6,l), Precond(1)\}
\end{aligned}$$

By treating *Precond* as fixed, we have obtained one model, in which the gun is loaded at time 5 and one, in which it is not loaded. These two models are representatives of two large equivalence classes of models, whose members differ in the extensions of *Holds*. Our equivalence class approach avoids storing all these models. The next reasoning step is filtering with the observations and the nochange axiom. We compute $Filter_{(Occlude,Precond,Precond')}(SCD,\{M_1,M_2\}, \bigwedge_{o \in Obs} o \wedge NCP)$. Both equivalence classes survive the filtering. The remaining representatives are shown below:



[1] $\oplus$ denotes the exclusive disjunction

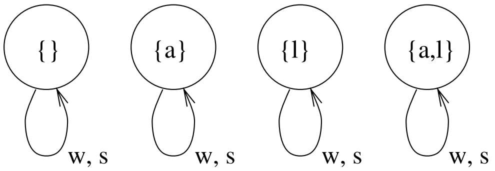
The two equivalence classes now only contain the two intended chronicle completions. If we ask whether the turkey is dead at time 6 by executing $Filter_{(Occlude,Precond,Precond')}(SCD \cup Obs \cup \{NCP\}, \{M_3, M_4\}, \neg Holds(6,a))$, model M_4 remains. As intended, the query cannot be proved and the system remains unspecific about the question, whether the turkey stays alive. On the other hand, if we ask whether the gun is unloaded at time 6, the system successfully proves this query (both models M_3 and M_4 are eliminated through filtering with $Holds(6,l)$).

3.5.2 Baker's Formalism

Baker proposed an approach for non–monotonic reasoning in the situation calculus [Bak91]. He extends previous approaches by introducing an existence of situations axiom, which guarantees the existence of a situation for every consistent combination of truth values of the fluents. Additionally Baker varies the *Result*–function instead of *Holds*. We implement Baker's approach using a language with three sorts: $\mathcal{A}$ for actions, $\mathcal{F}$ for fluents and $\mathcal{S} := 2^{\mathcal{F}}$ for situations, which are characterized by the fluents that hold. The usual *Result*–function of the situation calculus is replaced by a predicate *Result*, for which we postulate that it is functional by

$$\forall a \forall s \exists s' (Result(a,s,s') \wedge \forall s'' (Result(a,s,s'') \rightarrow s' = s''))$$

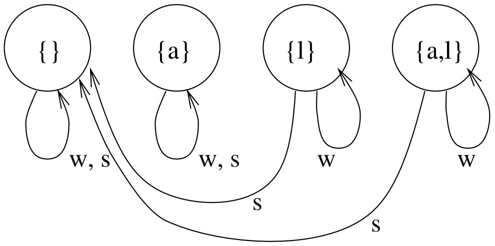
Consider the Yale Shooting Problem with fluents a (Alive) and l (Loaded) and the actions W (Wait) and S (Shoot). We start reasoning in a trivial model M_0 of the domain axioms without the description of the actions, i.e. in a completely inert world, where the actions lead to no change in the truth values of the fluents:



No abnormality has to be assumed because all fluents are allowed to persist. Now we revise this model with the axiom ACT of the shoot action ($Rev_{Ab}(\mathcal{S} \setminus \{ACT\}, M_0, \{ACT\})$). This axiom states that the turkey is dead and the gun is unloaded after shooting with a loaded gun.

$$\begin{aligned} ACT: \quad & \forall s\ (Holds(a,s) \wedge Holds(l,s) \wedge Result(S,s,s')) \\ & \rightarrow (\neg Holds(a,s') \wedge \neg Holds(l,s')) \end{aligned}$$

The following model M_1 is detected to be the minimal one.



Now we prove that the turkey is dead, when we start with a loaded gun and a living turkey then wait and then shoot.

$$\varphi : Result(W,\{a,l\},s_1) \wedge Result(S,s_1,s_2) \rightarrow \neg Holds(a,s_2)$$

We prove this by $Filter_{Ab}(\mathcal{S},\{M_1\},\neg\varphi)$, which returns $\emptyset$.

3.5.3 Kartha's Extension

Kartha [Kar94] recently reported that Baker's circumscription method is problematic in the presence of non–deterministic actions. He proposed to exclude the observations from the minimization in order to obtain the expected results. We will use our method to solve his two–buses example: A passenger has to buy a ticket before he can take a bus. After buying the ticket, he waits at the bus stop and gets on the first bus which arrives. This can be either the red or the yellow bus. The problem is described using the fluents H (has ticket), R (on red bus) and Y (on yellow bus) and the actions B (buy a ticket) and G (get on bus). A group C of axioms describes the dependencies among the fluents:

$$\begin{aligned} C: \quad & \forall s \neg(Holds(R,s) \wedge Holds(Y,s)) \\ & \forall s(Holds(R,s) \rightarrow Holds(H,s)) \\ & \forall s(Holds(Y,s) \rightarrow Holds(H,s)) \end{aligned}$$

Another group E of axioms describes the effect of the actions. Buying a ticket makes the fluent H true. Getting on the bus takes the passenger either on the red or the yellow bus, if he has a ticket. Note, that we again replace the *Result*–function by a functional predicate.

$$\begin{aligned} E: \quad & \forall s \forall s' \neg Holds(H,s) \wedge Result(B,s,s') \rightarrow Holds(H,s') \\ & \forall s \forall s' Holds(H,s) \wedge \neg Holds(R,s) \wedge \neg Holds(Y,s) \wedge Result(G,s,s') \rightarrow \\ & \qquad Holds(R,s') \vee Holds(Y,s') \end{aligned}$$

Additionally we have observed that buying a ticket and getting on a bus takes passengers on the red bus.

$$O: \quad \exists s_0 \forall s_1 \forall s_2 \neg Holds(H,s_0) \wedge Result(B,s_0,s) \wedge Result(G,s,s') \wedge Holds(R,s')$$

Since *Result* is functional, we can use either existential or universal quantification for the variables s and s' in the above sentence. Finally we use the usual frame axiom, adapted to our relational version of the *Result*–function.

$$F: \quad \forall f \forall a \forall s \forall s' (\neg Ab(f,a,s) \wedge Result(a,s,s') \rightarrow (Holds(f,s) \equiv Holds(f,s')))$$

Again, we use a sort $\mathcal{A}$ for the actions, a sort $\mathcal{F}$ for the fluents and $\mathcal{S} = 2^{\mathcal{F}}$ for the situations. However this time not all fluent combinations are consistent with the axioms. A predicate *Absit* is used to denote that a fluent combination is abnormal in the sense that it contradicts the axioms. We can express that all consistent fluent combinations exist, by the axiom *Abs*.

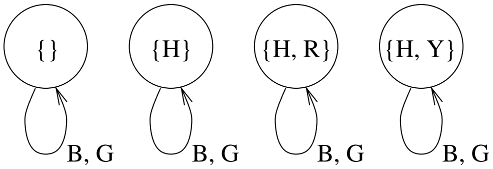
$$Abs: \quad \forall s(\neg Absit(s) \rightarrow \bigwedge_{f \in s} Holds(f,s))$$

Now we circumscribe *Absit* in the part of the theory, which makes statements about valid fluent combinations by executing $Rev_{Absit}(\emptyset, \{\emptyset\}, C \cup \{Abs\})$. The resulting model yields the following correct extension of *Absit*:

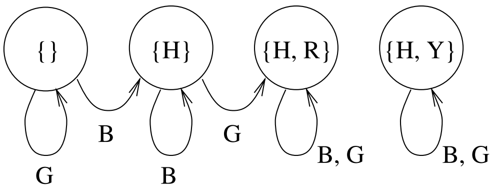
$$\{Absit(\{H,R,Y\}), Absit(\{R\}), Absit(\{R,Y\}), Absit(\{Y\})\}$$

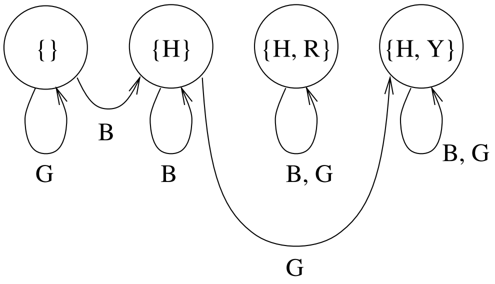
If we minimize *Absit* in a larger part of the theory (following Kartha, we must however exclude the observations), by executing $Rev_P(\emptyset, \{\emptyset\}, C \cup E \cup \{F\} \cup \{Abs\})$, our approach also returns one model with the correct extension. Here, the equivalence classes come into play again: We obtain one model with an arbitrary extension of *Result* and *Ab* representing a large class of models, where all models agree in the extension of *Absit*.

Now that we have found the consistent fluent combinations we proceed to the main problem, namely the computation of the correct models for the two buses problem. As for the yale shooting we again use a model of the static world as initial model and revise this model with the action axioms. The initial model M_0 is depicted below:



This model is now revised with the action axioms E by $Rev_{Ab}(C \cup \{F\} \cup \{Abs\}, \{M_0\}, E)$. To avoid unintended models, we must first exclude the observations from the minimization. The system returns the following two models: One, in which the passenger takes the red bus and one in which he takes the yellow bus.





Note, that the two models obtained are not $\sim^{Ab}$–equivalent, because in one model getting on the bus is abnormal wrt. the fluent Y and in the other this action is abnormal wrt. the fluent R. After the minimization of Ab we now have to filter using the observation O, by executing $Filter_{Ab}(C \cup E \cup \{F\} \cup \{Abs\}, \{M_1, M_2\}, O)$, which eliminates the model where G takes the passenger on the yellow bus.

3.5.4 Nixon's Diamond

Consider the following proof of concept example originally proposed by Reiter [RC81] and used by Ginsberg [Gin89]: US–President Nixon was both a republican and a quaker. Normally, republicans are hawks and quakers are no hawks. This is formalized by

$$\begin{aligned} \mathcal{S}: &\quad \forall x.Republican(x) \wedge \neg Ab(Political, x) \rightarrow Hawk(x) \\ &\quad \forall x.Quaker(x) \wedge \neg Ab(Religious, x) \rightarrow \neg Hawk(x) \\ Obs: &\quad Republican(Nixon) \\ &\quad Quaker(Nixon) \end{aligned}$$

First we compute the minimal models using $Rev_{Ab}(\emptyset, \{\emptyset\}, \mathcal{S} \cup Obs)$ and receive the two result models:

$$\begin{aligned} M_1: &\quad \{Ab(Political, Nixon), Republican(Nixon), Quaker(Nixon)\} \\ M_2: &\quad \{Ab(Religious, Nixon), Hawk(Nixon), Republican(Nixon), \\ &\quad Quaker(Nixon)\} \end{aligned}$$

Now we ask the system, if Nixon was a hawk by filtering with $\neg Hawk(Nixon)$. Model M_1 remains, indicating that this query cannot be proved.

Nixon's diamond has also been used as a proof of concept problem for prioritized circumscription in [WS97]. Instead of one Ab–predicate two different predicates are used: $Ab1(x)$ denotes that x is politically abnormal, $Ab2(x)$ denotes that x is religiously abnormal.

$$\begin{array}{ll} \mathcal{S}_P: & \forall x\, Republican(x) \land \neg Ab1(x) \rightarrow Hawk(x) \\ & \forall x\, Quaker(x) \land \neg Ab2(x) \rightarrow \neg Hawk(x) \end{array}$$

Our algorithm for circumscribing $\mathcal{S}_P$ in $Ab1 > Ab2$ first minimizes $Ab1$ (compare definition 3.12). The revision is $Rev_{Ab1}(\emptyset, \{\emptyset\}, \mathcal{S}_P \cup Obs)$. The result is the model

$$M : \{Ab2(Nixon), Hawk(Nixon), Republican(Nixon), Quaker(Nixon)\}.$$

Following definition 3.12 we must now minimize $Ab2$, while holding the previously minimized extension of $Ab1$ fixed. Thus, we compute

$$\begin{array}{ll} Rev_{Ab2}(& \{\forall x\, Republican(x) \land \neg Ab1(x) \rightarrow Hawk(x)\}, \\ & \{M\} \\ & \{\neg Ab1(Nixon), \forall x\, Quaker(x) \land \neg Ab2(x) \rightarrow \neg Hawk(x)\}) \end{array}$$

Again, the result is the same model M, as obtained from the first revision. If we ask the system if $Nixon$ was a hawk by executing $Filter_{(Ab1,Ab2)}(\mathcal{S}_P \cup Obs, \{M\}, \neg Hawk(Nixon))$, the result is $\emptyset$. Thus, $Hawk(Nixon)$ is proved. By assigning a higher priority to the minimization of political abnormality, we have given preference to the first rule, so that $Hawk(Nixon)$ can be inferred.

3.5.5 Running Times

The algorithms presented in this paper have been implemented in PROLOG. The following table shows the runtimes for the examples described above on a SPARC–Station 4.

Example	Runtime: Revision	Filtering	Total
PMON-Circumscription	0.14s	0.32s	0.46s
Yale Shooting (Baker)	0.23s	0.08s	0.34s
Absit-Minimization (Kartha)	3.28s	–	3.28s
Two Buses (Kartha)	0.29s	0.04s	0.33s
Nixon's Diamond	0.03s	0.01s	0.04s
Nixon's Diamond ($Ab1$, $Ab2$)	0.05s	0.02s	0.07s

In most examples the filtering step is executed much faster than the revision, because usually only few facts can be altered in the models without affecting the extension of the minimized predicate (which is not changed during filtering, see section 3.4.6). The PMON–example is an exception, because the equivalence classes of the minimal models are very large.

All examples except the *Absit*–minimization are solved in less than 0.5s. The longer runtime of the *Absit*–example is due to the structure of the axiom group C, which introduces a large amount of non–determinism into the revision. In this example we obtain a revision tree with many branches, which all consist of few repair steps.

3.6 Implementing Diagnosis with DRUM-II

In contrast to previous systems for model–based diagnosis which rely on techniques specific to diagnosis, DRUM-II handles diagnosis as a special case of circumscription. We will now relate the diagnosis definition from chapter 2 to the reasoning techniques presented previously in the current chapter.

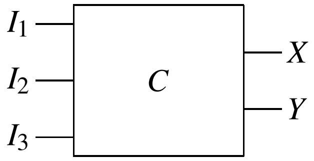
3.6.1 Consistency–Based Diagnosis with DRUM-II

Let us begin by establishing the connection between minimal consistency–based diagnosis and minimal model computation. As specified by definition 2.2 minimal diagnoses are minimal sets of *Ab*–atoms Δ, so that

$$SD \cup Obs \cup \{Ab(c) | c \in \Delta\} \cup \{\neg Ab(c) | c \in Comp \backslash \Delta\} \qquad (3.9)$$

is consistent. We want to characterize the same concept using models: If the theory specified in formula 3.9 is consistent, it has a model M. M has to interpret Ab exactly as specified by $\{Ab(c)|c \in \Delta\} \cup \{\neg Ab(c)|c \in Comp\backslash\Delta\}$, thus $M|Ab| = \Delta$. Consequently, for each minimal diagnosis of $SD \cup Obs$ there exists a minimal model M of $SD \cup Obs$, such that $M|Ab| = \Delta$. However it can be that several minimal models correspond to one diagnosis.

Example 3.38
Consider a system component C (an integrated digital circuit) with two output ports X and Y.



Suppose the system description predicts the values $X = 1$ and $Y = 0$, given the current values of I_1, I_2 and I_3. If we observe $Y = 1$, we have to assume that C is faulty. Since C is behaving abnormally (and no fault model has been specified for C), we cannot predict the value of X. Thus we have the following two $\leq^{Ab}$–minimal models corresponding to the minimal diagnosis $\{C\}$: $M_1 = \{Value(X,0), Value(Y,1), Ab(C)\}$ and $M_2 = \{Value(X,1), Value(Y,1), Ab(C)\}$. #

We can show that the minimal diagnoses directly correspond to the $\sim^{Ab}$-equivalence classes of the $\leq^{Ab}$–minimal models.

Theorem 3.39 *Let $\mathcal{M}$ be a transversal of $MOD^{Ab}(SD \cup Obs)/\sim^{Ab}$. Then $\mathcal{D} := \{M|Ab| \; |M \in \mathcal{M}'\}$ is the set of all minimal diagnoses of $(SD, Comp, Obs)$.*

Proof: First we show that every $\Delta \in \mathcal{D}$ is indeed a minimal diagnosis of $(SD, Comp, Obs)$. Since $\Delta \in \mathcal{D}$, there exists a $\leq^{Ab}$–minimal model M of $SD \cup Obs$, so that $M|Ab| = \Delta$. This M is also a model of $SD \cup Obs \cup \{Ab(c)|c \in M|Ab|\} \cup \{\neg Ab(c)|c \in Comp \backslash M|Ab|\}$. Since $\Delta = M|Ab|$ we can write this theory as $SD \cup Obs \cup \{Ab(c)|c \in \Delta\} \cup \{\neg Ab(c)|c \in Comp \backslash \Delta\}$. Since M is a model of this theory, the theory is logically consistent and thus Δ is a diagnosis.

We can prove by contradiction, that Δ is also minimal. Assume a diagnosis Δ' exists, so that $\Delta' \subset \Delta$. Then $SD \cup Obs \cup \{Ab(c)|c \in \Delta'\} \cup \{\neg Ab(c)|c \in Comp \backslash \Delta'\}$ is consistent. Let M' be a model of this theory. Then, M is also a model of $SD \cup Obs$. Furthermore, we know that $M|Ab| = \Delta'^{2}$. Consequently, M' is a model of $SD \cup Obs$ with $M'|Ab| \subset M|Ab|$. This contradicts the assumption that M is a minimal model of $SD \cup Obs$.

Now we show that every minimal diagnosis Δ is contained in $\mathcal{D}$. If Δ is a minimal diagnosis, then $SD \cup Obs \cup \{Ab(c)|c \in \Delta\} \cup \{\neg Ab(c)|c \in Comp \backslash \Delta\}$ is logically consistent and thus has at least one model M. Let us finally prove by contradiction, that M is $\leq^{Ab}$–minimal. Assume, that a model M' of $SD \cup Obs$ exists, so that $M' \leq^{Ab} M$. M' is also a model of $SD \cup Obs \cup \{Ab(c)|c \in M'|Ab|\} \cup \{\neg Ab(c)|c \in Comp \backslash M'|Ab|\}$. Thus, $M'|Ab|$ is a diagnosis[3] so that $M'|Ab| \subset M|Ab| = \Delta$. This contradicts our assumption that Δ is a minimal diagnosis. Q.E.D.

While previous results on the relation of diagnosis and circumscription [BC94, Rai90] focus on formalizing stronger forms of explanation (than provided by consistency–based diagnosis), this theorem establishes an interesting connection between consistency–based diagnosis and circumscription, because $\mathcal{M}$ is the set of all models obtained by circumscribing $SD \cup OBS$ in Ab, while varying all other predicates.

3.6.2 Computing Spectrum Diagnoses with DRUM–II

The definition of a spectrum diagnosis (definition 2.18) adds an abductive element to the consistency–based definition used in the previous sections of this chapter. If multiple fault models are used, detailed information about the fault models must now be contained in every diagnosis.

Example 3.40 As a minimal example of the need for detailed information about the fault models consider an inverter gate with the fault models stuck–at–0 ($S0$) and stuck-at-1 ($S1$) and the following system description:

$$SD: \quad Ok(Inv) \rightarrow (High(Inv, O) \Leftrightarrow \neg High(Inv, I)) \tag{3.10}$$

[2] Note however, that this is not a logical consequence. The model could contain an atom $Ab(x)$ so that $x \notin Comp$. However, it is an implicit assumption throughout the diagnosis literature, that Ab has only components as arguments

[3] Assuming again, that Ab has only components as arguments

$$Mode(Inv, S0) \rightarrow \neg High(Inv, O) \tag{3.11}$$

$$Mode(Inv, S1) \rightarrow High(Inv, O) \tag{3.12}$$

$$\forall c\ (Ab(c) \Leftrightarrow \exists m\, Mode(c, m) \wedge m \neq Ok) \tag{3.13}$$

Axiom 3.10 denotes that the voltage at the output of a correctly working inverter is the negation of the voltage at the input. Axioms 3.11 and 3.12 define the fault models: stuck-at-1 means high voltage at the output while stuck-at-0 means low voltage at the output. Axiom 3.13 denotes that abnormality is equivalent to a mode other than *Ok*. It was motivated in section 2.3. Suppose we observe low voltage both at the input and at the output and we want the system to explain the low voltage at the output:

$$\begin{aligned} Obs_{In}: &\ \neg High(Inv, I) \\ Obs_{Out}: &\ \neg High(Inv, O) \\ Obs^{+}: &\ \neg High(Inv, O) \end{aligned}$$

The diagnosis (à la Reiter) $\{Ab(Inv)\}$ is not sufficient to explain the output, because the inverter could be either in mode $S0$ or $S1$ and only one of them explains the observation. The mode assignment $\{Mode(Inv, S0)\}$ is however sufficient to logically explain the observation, because the low voltage can be inferred using axiom 3.11.

Thus we need to compute all minimal mode assignments. Remember, that we define $\forall c\ (Ab(c) \equiv \exists m\ (Mode(c, m) \wedge m \neq Ok))$ when working with mode assignments, so that we can still minimize the *Ab*-predicate. We now first minimize the *Ab*–predicate and afterwards maintain every consistent extension of the *Mode*–predicate by minimizing *Mode* and its negation $\overline{Mode}$ in parallel as described in section 3.3.1.

Definition 3.41 MA_{min}
Let $(SD, Comp, Obs = Obs_{in} \dot{\cup} Obs_{Out}, Obs^{+})$ *be a diagnosis problem and* M *a model for* $SD \cup Obs_{In}$*.*

$$MA_{min}(SD, M, Obs) := Rev_{Ab > (Mode, \overline{Mode})}(SD, M, Obs)$$

where $\overline{Mode}(c, m) \Leftrightarrow \neg Mode(c, m)$*.*

The mode assignment corresponding to a model M is the set of all *Mode*–atoms true in M. It is denoted by $MA(M)$.

Definition 3.42 $MA(M)$.
Let M *be a model.*

$$MA(M) := \{Mode(c, m) | Mode(c, m) \in M\}$$

The following proposition shows that MA_{min} actually computes a model for each minimal mode assignment.

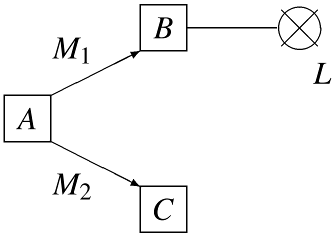
Proposition 3.43 *Let $(SD, Comp, Obs = Obs_{in} \dot{\cup} Obs_{Out}, Obs^{+})$ be a diagnosis problem and M a model for $SD \cup Obs_{In}$. Let MA be the set of all minimal mode assignments D such that $SD \cup Obs \cup D$ is consistent.*

Then, for each minimal mode assignment D, there is a model $M \in MA_{min}(SD, \{M\}, Obs)$, such that $D = MA(M)$.

Proof Sketch: $MA_{min}(SD, M, Obs) = Rev_{Ab>(Mode,\overline{Mode})}(SD, M, Obs)$ yields a transversal of $MOD^{Ab>(Mode,\overline{Mode})}(SD \cup Obs)$. This transversal contains a model for every minimal extension of Ab and every extension of $Mode$. If now D is a minimal mode assignment, it corresponds to a model M_D with a minimal extension of Ab. Since the extension of Ab is minimal in M_D, we know that there is a suitable model M in the transversal, which agrees with M_D in the extensions of Ab and $Mode$. Q.E.D.

Although MA_{min} is the set of all minimal mode assignments, it does not yet contain all spectrum diagnoses. The following example shows that there are minimal spectrum diagnoses, which do not correspond to $\leq^{Ab}$–minimal models of $SD \cup Obs$.

Example 3.44 Consider a computer network consisting of three computers A, B, and C. Computer A sends a message to both B and C using an unreliable protocol. Computer B is connected to a status lamp, which is lit if B is faulty.



The system description SD consists of the following formulas:

$$\forall c_1 \forall m \forall c_2 \; Sent(c_1, m, c_2) \wedge Mode(c_2, Ok) \rightarrow Recd(c_2, m) \tag{3.14}$$

$$Mode(A, Ok) \rightarrow (Sent(A, M_1, B) \wedge Sent(A, M_2, C)) \tag{3.15}$$

$$Mode(B, Ab) \rightarrow Lit(L) \tag{3.16}$$

Formula 3.14 denotes that a message m sent from a computer c_1 to a computer c_2 will arrive at c_2 unless c_2 is faulty. Formula 3.15 states that A sends messages to B and C if it is working properly. Formula 3.16 denotes that lamp L is lit upon failure of B. Let us assume that the message neither arrived at B nor at C and that the lamp L is lit:

$$Obs := Obs_{Out} := \{\neg Recd(B, M_1), \neg Recd(C, M_2), Lit(L)\} \tag{3.17}$$

We want the system to explain that L is lit:

$$Obs^{+} := \{Lit(L)\} \tag{3.18}$$

There are two $\leq^{Ab}$–minimal models of $SD \cup Obs$, namely

$$\begin{aligned} M_1 &: \{Sent(A,M_1,B), Sent(A,M_2,B), Lit(B), Mode(A,Ok), \\ &\quad Mode(B,Ab), Mode(C,Ab), Ok(A), Ab(B), Ab(C)\} \qquad (3.19) \\ M_2 &: \{Mode(A,Ab), Mode(B,Ok), Mode(C,Ok), \\ &\quad Ab(A), Ok(B), Ok(C), Lit(L)\} \qquad (3.20) \end{aligned}$$

Model M_2 shows that computing minimal models provides only a weak form of explanation. The lamp L is lit but there is no explanation for this in the model. Following Console and Torasso [CT91] one can say, that the mode assignment $\{Mode(A,Ab), Mode(B,Ok), Mode(C,Ok)\}$ corresponding to the model M_2 assumes an anonymous cause for the lamp's being lit, which is not modeled in *SD*. Let us now check if the minimal models M_1 and M_2 correspond to spectrum diagnoses. Since both models are consistent with $SD \cup Obs$, we only have to check, whether Obs^+ is entailed. $M_1 \models Obs^+$, since $Mode(B,Ab) \in M_1$ and $Mode(B,Ab) \models Lit(L)$. Thus, M_1 corresponds to the spectrum diagnosis $D_1 = \{Mode(A,Ok), Mode(B,Ab), Mode(C,Ab)\}$.

The second model does not entail Obs^+, since the only possible explanation $Mode(B,Ab)$ is missing. There is however a second minimal spectrum diagnosis $\Delta_2 = \{Mode(A,Ab), Mode(B,Ab), Mode(C,Ok)\}$ corresponding to the model $M_3 = (M_2 \setminus \{Mode(B,Ok)\}) \cup \{Mode(B,Ab), Ab(B)\}$, which is obviously not $\leq^{Ab}$–minimal, because $M_3|Ab| \supset M_2|Ab|$. #

This example has demonstrated that spectrum diagnoses do not directly correspond to minimal models. The reason is that a non–minimal model may be necessary to explain (i.e. logically entail) the observations Obs^+ in the abductive part of the diagnosis definition. In general, the presence of a set of observations Obs^+ which have to be abductively explained makes computing diagnoses much harder. On purely consistency–based problems the search for diagnoses is guided by the observations: DRUM–II inserts the observations into a model of the correctly functioning system. It restores consistency by applying local changes to the model until consistency is restored. During this repair process DRUM–II only considers a small subset of the components as diagnoses. In abduction problems the observations have to be logically entailed. They cannot be inserted into the models to guide the search.

The definition of a spectrum diagnosis comprises a consistency–based part (the system description has to be consistent with all observations) and an abductive part (the observations in Obs^+ have to be entailed). Since DRUM–II implements consistency–based reasoning very efficiently our algorithm for computing minimal spectrum diagnosis first searches for minimal consistency–based diagnoses. For each minimal diagnosis it checks, if the observations in Obs^+ are entailed. The diagnoses which entail Obs^+ are minimal spectrum diagnoses.

Additionally, there may be supersets of minimal consistency–based diagnoses, which are minimal spectrum diagnoses. We can find these by modifying the system

Algorithm 3.45 *Compute all Minimal Spectrum Diagnoses*

```
FUNCTION Spectrum($SD$, $Comp$, $Obs = Obs_{In} \dot{\cup} Obs_{Out}$, $Obs^+$)
$Diags$ := $\emptyset$;
REPEAT
    $\mathcal{D}$ := ``All minimal mode assignments $D$
                        s.th. $SD \cup Obs \cup D \not\models \bot$'';
    $\mathcal{D}^{Min}$  := $\{D \in \mathcal{D} | \nexists D' \in Diags : D' \leq^{Ab} D\}$;
    $\mathcal{D}^{Min}_{\models}$  := $\{D \in \mathcal{D} | SD \cup Obs_{In} \cup D \models Obs^+\}$;
    $\mathcal{D}^{Min}_{\not\models}$  := $\mathcal{D}^{Min} \setminus \mathcal{D}^{Min}_{\models}$;
    $Diags$  := $Diags \cup \mathcal{D}^{Min}_{\models}$;
    $SD$  := $SD \cup \bigcup_{D \in \mathcal{D}^{Min}_{\not\models}} \left\{ \bigvee_{c \in Comp: Mode(c,Ok) \in D} Ab(c) \right\}$;
UNTIL $\mathcal{D}^{Min}_{\not\models} = \emptyset$;
RETURN $Diags$;
```

description. Consider a system with components $Comp = \{A, B, C, D, E\}$ and a $\leq^{Ab}$–minimal mode assignment $\{Mode(A, Ab), Mode(B, Ab), Mode(C, Ok), Mode(D, Ok), Mode(E, Ok)\}$, which is not a spectrum diagnosis. Now possibly a superset of this diagnosis is a minimal spectrum diagnosis. We can turn these supersets into minimal mode assignments by extending the system description: $SD' := SD \cup \{Ab(C) \vee Ab(D) \vee Ab(E)\}$. The new sentence is formally a conflict. It makes the diagnosis $\{Mode(A, Ab), Mode(B, Ab), Mode(C, Ok), Mode(D, Ok), Mode(E, Ok)\}$ inconsistent, because this diagnosis entails $\neg Ab(C) \wedge \neg Ab(D) \wedge \neg Ab(E)$ which is the negation of the new sentence $Ab(C) \vee Ab(D) \vee Ab(E)$ in SD'. The algorithm iterates the extension of SD and computation of diagnoses until no new minimal diagnoses not entailing Obs^+ are found. This basic algorithm is summarized in algorithm 3.45.

Algorithm 3.45 computes all diagnoses of the new system description in every iteration. Now consider the implementation of this algorithm using the DRUM–II functions MA_{min} and $Filter$ (algorithm 3.46). In this implementation we compute diagnoses more selectively. The algorithm works as follows: First, it computes all minimal mode assignments for the system description together with the observations. It partitions the mode assignments into a set $\mathcal{M}^{Min}_{\models}$ of mode assignments, which entail the observations in Obs^+ and a set $\mathcal{M}^{Min}_{\not\models}$ of mode assignments, which do not entail the observations in Obs^+. The mode assignments corresponding to the models in $\mathcal{M}^{Min}_{\models}$ are added to the set of diagnoses ($Diags$). As we motivated in example 3.44, models assuming more abnormality than those in $\mathcal{M}^{Min}_{\not\models}$ can still correspond to minimal spectrum diagnoses. Therefore, a clause set New is added to the system description, which makes sure, that none of the models in $\mathcal{M}^{Min}_{\not\models}$ is a model of $SD \cup New$, so that mod-

Algorithm 3.46 *Compute all Minimal Spectrum Diagnoses with DRUM–II*

```
FUNCTION Spectrum′($SD$, $Comp$, $Obs = Obs_{In} \dot{\cup} Obs_{Out}$, $Obs^+$, $M$)
$Diags$ := $\emptyset$;
$\mathcal{M}^{Min}$ := $MA_{min}(SD \cup Obs_{In}, \{M\}, Obs_{Out})$;
REPEAT
    $\mathcal{M}^{Min}_{\models}$ := $\{M \in \mathcal{M}^{Min} |$
                $Filter_{Mode}(SD \cup Obs_{In}, \{M\}, \{\{ \bigvee_{o \in Obs^+} \neg o\}\}) = \emptyset\}$;
    $\mathcal{M}^{Min}_{\not\models}$ := $\mathcal{M}^{Min} \setminus \mathcal{M}^{Min}_{\models}$;
    $Diags$ := $Diags \cup \bigcup_{M \in \mathcal{M}^{Min}_{\models}} MA(M)$;
    IF $\mathcal{M}^{Min}_{\not\models} \neq \emptyset$ THEN
        $New$ := $\left\{ \{ \bigvee_{c \in Comp : Mode(c,Ok) \in MA(M)} Ab(c)\} | M \in \mathcal{M}^{Min}_{\not\models} \right\}$;
        $Ab_M$ := $\{Ab(c) | c \in Comp$ and $Mode(c, Ok) \notin M\}$;
        $\mathcal{M}$ := $\bigcup_{M \in \mathcal{M}^{Min}_{\not\models}} MA_{min}(SD \cup Obs \cup Ab_M, \{M\}, New)$;
        $\mathcal{M}^{Min}$ := $\{M \in \mathcal{M} | \nexists D \in Diags : D \leq^{Ab} MA(M)\}$;
        $SD$ := $SD \cup New$;
    END IF;
UNTIL $\mathcal{M}^{Min}_{\not\models} = \emptyset$;
RETURN $Diags$;
```

els with more abnormality are considered in the next iteration. However, in this next iteration the optimized algorithm only revises the models from $M^{Min}_{\not\models}$, in contrast to algorithm 3.45, which recomputes all minimal mode assignments. The REPEAT-UNTIL loop terminates, when no new minimal mode assignments are found. In practice, we can define an upper bound on the number of faults in a diagnosis to speed up computation.

A formal detail of algorithm 3.46 is the set Ab_M. It is needed, because the revision function used in the definition of MA_{Min} is only defined, if the set of models in the second parameter is a transversal of the minimal models of the theory in the first parameter (compare definition 3.5). This precondition is trivially satisfied, because we added the set Ab_M, consisting of all the abnormals in M, to the theory for this revision.

3.7 Discussion

The first DRUM system [Ned93, NG94] was strongly influenced by the ideas of Chou and Winslett's model–based belief revision system IMMORTAL [CW94]. DRUM used optimized versions of the minimal change algorithms by Chou and Winslett and applied an additional minimization step to obtain only minimal extensions of the *Ab*-predicate.

As we have described in this chapter, DRUM–II replaces the minimal change semantics completely by the semantic approach to circumscription. It provides a very general implementation of circumscription for fixed domain theories, covering parallel and prioritized circumscription, fixed and variable predicates. Our implementations of spectrum diagnosis and several formalism from non–monotonic reasoning show that this expressive power is needed to handle current AI formalisms.

Furthermore, the search for models is much more efficient in DRUM–II than in DRUM and IMMORTAL. The use of an iterative deepening algorithm increases the efficiency of DRUM–II dramatically, because it avoids unnecessary copying of models. Furthermore, DRUM–II uses a simpler definition of a flipping than both DRUM and IMMORTAL: While the previous systems use hitting sets over the set of violated clauses as flippings, DRUM–II uses only single literals. Our critical examination of hitting sets as flippings [NF96] has shown that they lead to an unnecessary combinatorial explosion of the number of flippings in many situations.

Chapter 4

Circuit–Diagnosis with DRUM–II

The diagnosis of digital circuits is the classical application of consistency–based diagnosis. Circuits have been studied in the very first papers as proof of concept problems. However, it took several years of research before efficient variants of ATMS–based diagnosers were capable of solving large combinatorial circuits [dK91, RdKS93]. A set of combinatorial circuits distributed at the 1985 symposium on circuits and systems [Isc85] has served different researchers as a benchmark for evaluating the efficiency of diagnostic systems.

We start this chapter with an overview of circuit modeling and diagnosis in DRUM–II using a parity checker as the running example. We identify potential for optimization in the basic DRUM–II algorithm presented in the previous chapter. Then, we extend our work in [NF96, FN97] and introduce a variant of the algorithm, which focuses the search for diagnoses by exploiting the structure of the device under consideration. In contrast to ATMS–based systems, which record information during search, our algorithm exploits precompiled information. While dependency–recording at runtime leads to combinatorial explosion of the data–structures used, the size of the precompiled information used by the optimized DRUM–II engine is only quadratic in the size of the device.

After introducing the optimized algorithm we provide a characterization of the abovementioned benchmark circuits. In particular, we discuss why the reconvergent fanout of these circuits makes them hard to solve for model–based diagnosis engines. Finally, we present the running times of the optimized DRUM–II on these benchmark examples and compare them to other recent results. It turns out that DRUM–II performs much better than all previous systems on nearly all examples.

4.1 Diagnosing Digital Circuits at Gate Level

To provide an overview of circuit diagnosis in DRUM–II we will consider the diagnosis of a 9–bit parity checker taken from the Texas Instruments SN74LS280 chip. This circuit was already considered in [Out93]. We will start this section by discussing the

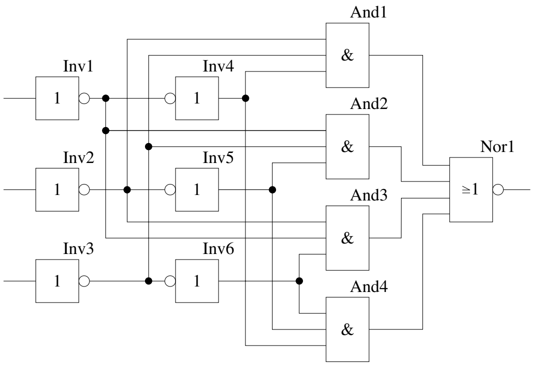


Figure 4.1: A 3-bit Parity Checker

system description and execution trace of DRUM–II for the smaller three bit parity checker shown in figure 4.1 (which is part of the 9–bit parity checker) and move to the full 9–bit circuit later for the discussion of multiple faults and efficiency.

4.1.1 System Description

The system description of the circuit is mainly based on the following predicates:

$Val(c,p,v)$ denotes that the value of port p of component c has the value v. In digital circuits we use the values 0 (low voltage) and 1 (high voltage) and name the ports O (output of a component) and I_k (k-th input of a component).

$Conn(c_1,p_1,c_2,p_2)$ denotes that the port p_1 of a component c_1 is connected to the port p_2 of a component c_2.

$Type(c,t)$ denotes that c is a component of type t. In the parity checker the component types are *Inv* (Inverter), *And*3 (And–Gate with 3 inputs) and *Nor*4 (Nor–Gate with 4 inputs).

$TypePort(t,p)$ denotes that components of type t have a port called p. For example, for an inverter the facts $TypePort(Inv,I_1)$ and $TypePort(Inv,O)$ are true.

Now let us consider the axioms of the system description. Formula 4.1 denotes that the value of a port p_1 of a component c_1 has the same value as port p_2 of component c_2, if these ports are connected.

$$\begin{array}{l}\forall c_1 \forall c_2 \forall p_1 \forall p_2 \forall v \\ \quad Conn(c_1,p_1,c_2,p_2) \rightarrow (Val(c_1,p_1,v) \leftrightarrow Val(c_2,p_2,v))\end{array} \tag{4.1}$$

Next, we discuss the axioms for the different component types. The behavior of an inverter is described by formulas 4.2 and 4.3. If the inverter is working according to its specification ($\neg Ab(c)$) the output is the logical negation of the input.

$$\forall c \quad (Type(c,Inv) \wedge \neg Ab(c)) \rightarrow (Val(c,I_1,0) \rightarrow Val(c,O,1)) \tag{4.2}$$

$$\forall c \quad (Type(c,Inv) \wedge \neg Ab(c)) \rightarrow (Val(c,I_1,1) \rightarrow Val(c,O,0)) \tag{4.3}$$

Similarly, axioms 4.4 and 4.5 describe the correct behavior of an and–gate with three inputs. If the value of all three inputs is 1, the output is also 1. If one of the inputs is 0, the output is 0.

$$\begin{aligned}&\forall c\ (Type(c,And3) \wedge \neg Ab(c)) \rightarrow \\ &\quad ((Val(c,I_1,1) \wedge Val(c,I_2,1) \wedge Val(c,I_3,1)) \rightarrow Val(c,O,1))\end{aligned} \tag{4.4}$$

$$\begin{aligned}&\forall c\ (Type(c,And3) \wedge \neg Ab(c)) \rightarrow \\ &\quad ((Val(c,I_1,0) \vee Val(c,I_2,0) \vee Val(c,I_3,0) \rightarrow Val(c,O,0))\end{aligned} \tag{4.5}$$

The behavior of a nor–gate is formalized by axioms 4.6 and 4.7, which state that its output is 1, if all inputs are 0, and otherwise the output is 0.

$$\begin{aligned}&\forall c\ (Type(c,Nor4) \wedge \neg Ab(c)) \rightarrow \\ &\quad ((Val(c,I_1,0) \wedge Val(c,I_2,0) \wedge Val(c,I_3,0) \wedge Val(c,I_4,0)) \rightarrow Val(c,O,1))\end{aligned} \tag{4.6}$$

$$\begin{aligned}&\forall c\ (Type(c,Nor4) \wedge \neg Ab(c)) \rightarrow \\ &\quad ((Val(c,I_1,1) \vee Val(c,I_2,1) \vee Val(c,I_3,1) \vee Val(c,I_4,1)) \rightarrow Val(c,O,1))\end{aligned} \tag{4.7}$$

We need one additional axiom for the computation of diagnoses (formula 4.8), which states that every port in the circuit must have a value of 1 or 0. This axiom would however decrease the efficiency of the initial model generation. Postulating that every port has a value, it would force DRUM–II to assume arbitrary values for every port of the circuit at the beginning of the model generation, which would lead to a large number of inconsistent models. Therefore the axiom is qualified by a proposition *DiagnosisPhase*, which is false during model generation and true during diagnosis.

$$\begin{aligned}&DiagnosisPhase \rightarrow (\forall c \forall t \forall p\ (Type(c,t) \wedge TypePort(t,p)) \rightarrow \\ &\qquad ((Val(c,p,1) \leftrightarrow \neg Val(c,p,0))))\end{aligned} \tag{4.8}$$

The above axioms can be used for any circuit involving these components and are independent of the particular structure of circuit under consideration. The structure of the given circuit is modeled separately by a set of facts. First we need facts describing the types of the gates.

$Type(Inv1,Inv).\ Type(Inv2,Inv).\ Type(Inv3,Inv).\ Type(Inv4,Inv).$
$Type(Inv5,Inv).\ Type(Inv6,Inv).\ Type(And1,And3).\ Type(And2,And3).$
$Type(And3,And3).\ Type(And4,And3).\ Type(Nor1,Nor4).$

For axiom 4.8 we also need facts describing the ports of the component types used.

$$
\begin{aligned}
&TypePort(Inv,I_1).\ TypePort(Inv,O).\\
&TypePort(And3,I_1).\ TypePort(And3,I_2).\ TypePort(And3,I_3).\\
&TypePort(And3,O).\\
&TypePort(Nor4,I_1).\ TypePort(Nor4,I_2).\ TypePort(Nor4,I_3).\\
&TypePort(Nor4,I_4).\ TypePort(Nor4,O).
\end{aligned}
$$

Finally, we need a set of facts describing the connections among the components.

$$
\begin{aligned}
&Conn(Inv1,O,Inv4,I_1).\ Conn(Inv1,O,And2,I_1).\ Conn(Inv1,O,And3,I_2).\\
&Conn(Inv2,O,And1,I_1).\ Conn(Inv2,O,Inv5,I_1).\ Conn(Inv2,O,And3,I_1).\\
&Conn(Inv3,O,And1,I_2).\ Conn(Inv3,O,And2,I_2).\ Conn(Inv3,O,Inv6,I_1).
\end{aligned}
$$

$$
\begin{aligned}
&Conn(Inv4,O,And1,I_3).\ Conn(Inv4,O,And4,I_3).\\
&Conn(Inv5,O,And2,I_3).\ Conn(Inv5,O,And4,I_2).\\
&Conn(Inv6,O,And3,I_3).\ Conn(Inv6,O,And4,I_1).
\end{aligned}
$$

$$
\begin{aligned}
&Conn(And1,O,Nor1,I_1).\ Conn(And2,O,Nor1,I_2).\\
&Conn(And3,O,Nor1,I_3).\ Conn(And4,O,Nor1,I_4).
\end{aligned}
$$

4.1.2 Generating the Initial Model

DRUM–II uses the system description consisting of the above axioms and facts both for the computation of the initial model of the correctly functioning device and for the diagnosis. In the model generation phase, the values are subsequently propagated through the circuit. The number of steps needed by DRUM–II for this propagation is equal to the number of ports, for which values are computed. Thus, there is no need for a specialized simulator here. Let us assume that the input of the first inverter is 1 and the inputs of $Inv2$ and $Inv3$ are both 0.

$$Obs_{In} := \{Val(Inv1,I_1,1), Val(Inv2,I_1,0), Val(Inv3,I_1,0)\}$$

Following chapter 3, we can now generate the initial model by executing the revision $Rev_{Ab}(SD,\emptyset,Obs_{In})$. Let us study a part of the trace of this revision.

```
Iterative Deepening at Level 0
-----------------------------------

|| Level: 1
|| Step:  [val(inv1, i1, 1), val(inv2, i1, 0),
           val(inv3, i1, 0)]
```

DRUM–II starts by announcing that it will set the size of the diagnoses to ≤ 0 (compare with algorithm 3.32). Thus, it first tries to find a model of the system description without assuming faults. The output `Level: 1` means that DRUM–II is making the first change to the model, i.e. Level is the current depth of the search tree. DRUM–II inserts the input observations into the model. After `Step:` it outputs the literals it is going to flip.

```
|| Level: 2
|| Step:  [val(inv1, o, 0)]
```

The insertion of the input observations has lead to a violation of axiom 4.3: The model contains the fact $Val(Inv1, I_1, 1)$, the inverter $Inv1$ is not considered faulty, but no output value is assumed. To restore consistency, the value of the output of $Inv2$ is set to 0 in step 2. However, the output of the inverter $Inv1$ is connected to the input of the inverter $Inv4$. Consequently, DRUM–II concludes in step 3 that the input of $Inv4$ has the value 0.

```
|| Level: 3
|| Step:  [val(inv4, i1, 0)]
```

In this way, values are propagated through the circuit.

```
|| Level: 4
|| Step:  [val(inv4, o, 1)]
|| Level: 5
|| Step:  [val(and1, i3, 1)]
|| Level: 6
|| Step:  [val(and4, i3, 1)]
...
```

After 33 such steps the solution is found, because no further inconsistency arises. The model found by DRUM–II is shown in figure 4.2. This model correctly predicts an output of 0 for the parity checker, which is correct because of the odd parity of the input pattern.

4.1.3 Computing Diagnoses

Now, suppose we observe the value 1 at the output of the parity checker. We can compute the diagnoses for this symptom by executing the revision $Rev_{Ab}(SD \cup Obs_{In}, M_0, \{Val(Nor1, O, 1)\})$, where M_0 is the initial model depicted in figure 4.2.

DRUM–II starts the diagnosis trying to find a model with 0 *Ab*–facts. Obviously, this is not possible because the output observations contradict the input observations. DRUM–II detects this after 49 flippings and sets the diagnosis size to 1.

```
Iterative Deepening at Level 1
----------------------------------

|| Level: 1
|| Step:  [diagnosis_phase(t), val(nor1, o, 1)]
```

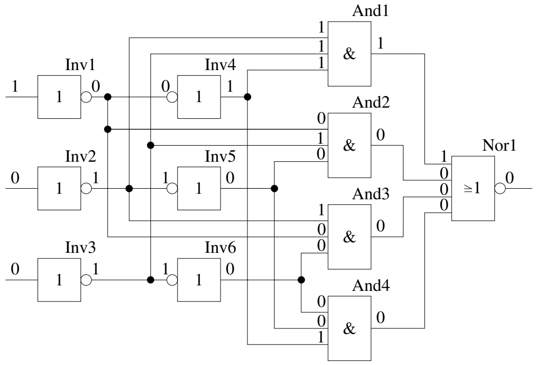


Figure 4.2: Initial model for the 3 Bit Parity Checker

In the first revision step shown above two atoms are integrated into the model: the output observation $Val(Nor1, O, 1)$ and the previously mentioned proposition *DiagnosisPhase*, which forces each port to assume either the value 0 or 1 but not both.

In contrast to the model generation phase, where values were propagated from the inputs to the outputs of the circuit, the incorporation of a symptom at the outputs leads to propagation of changes from the outputs to the inputs.

```
|| Level: 2
|| Step:  [not(val(nor1, o, 0))]
```

Due to the axiom 4.8 every propagation consists of two flippings. In steps 1 and 2 the output value of the nor–gate was inverted. Step 1 inserted the new value 1 for this port. Step 2 deleted the value 0 for the same port, because after axiom 4.8 a port can have at most one value at a time.

```
|| Level: 3
|| Step:  [not(val(nor1, i1, 1))]
|| Level: 4
|| Step:  [val(nor1, i1, 0)]
```

The explanation for flippings 3 and 4 is similar: Since the output of the nor–gate was changed to 1 in step 1 it is now inconsistent that the first input has value 1. Thus, this input value deleted from the model in step 3. Again, axiom 4.8 comes into play; it postulates that the first input of the nor–gate must have some value and therefore 0 is assumed. At this point, recall that DRUM–II avoids cycles during search by inverting the truth value of a literal at most once on each branch of the search tree. Therefore, DRUM–II does not consider satisfying axiom 4.8 by assuming $Val(Nor1, I_1, 1)$ again.

The next propagation steps follow the pattern described above. Changes are subsequently propagated in the direction of the inputs.

```
|| Level: 5
|| Step:  [val(and1, o, 0)]
|| Level: 6
|| Step:  [not(val(and1, o, 1))]
|| Level: 7
|| Step:  [not(val(and1, i2, 1))]
|| Level: 8
|| Step:  [val(and1, i2, 0)]
|| Level: 9
|| Step:  [val(inv3, o, 0)]
|| Level: 10
|| Step:  [not(val(inv3, o, 1))]
|| Level: 11
|| Step:  [not(val(and2, i2, 1))]
|| Level: 12
|| Step:  [val(and2, i2, 0)]
|| Level: 13
|| Step:  [not(val(inv6, i1, 1))]
|| Level: 14
|| Step:  [val(inv6, i1, 0)]
|| Level: 15
|| Step:  [val(inv6, o, 1)]
|| Level: 16
|| Step:  [not(val(inv6, o, 0))]
|| Level: 17
|| Step:  [not(val(and3, i3, 0))]
|| Level: 18
|| Step:  [val(and3, i3, 1)]
|| Level: 19
|| Step:  [not(val(and4, i1, 0))]
|| Level: 20
|| Step:  [val(and4, i1, 1)]
Assuming Abs[ab(inv3)]([]) after 70 Flippings
|| Level: 21
|| Step:  [ab(inv3)]
 ... Solution (after 71 Flippings)
Saving Model 2
```

On level 9 and 10 in the above trace DRUM–II changed the output value of the inverter *Inv*3 from 1 to 0. After repairing some other inconsistencies in the steps 11 to 20 DRUM–II reconsiders this inverter. However, further propagation of changes is impossible because the input value of *Inv*3 is an observation. The only way to restore consistency is assuming that the inverter is behaving abnormally, because both its input and output are 0. The first diagnosis is found and indicated by the output `Solution`.

DRUM–II tries to find further solutions by backtracking. The propagation described above has created many choice points for backtracking. This is because all the component axioms have an *Ab*–literal in their precondition. Thus, they can be satisfied either by the propagation of values or by assuming that the component is abnormal.

```
Assuming Abs[ab(inv6)]([]) after 71 Flippings
|| Level: 15
|| Step:  [ab(inv6)]
Assuming Abs[ab(inv3)]([ab(inv6)]) after 72 Flippings
```

Backtracking leads DRUM–II to assume that inverter *Inv*6 is faulty. However this assumption does not resolve the violation of the correct behavior of *Inv*3 whose input and output both have the value 0. Thus, DRUM–II must assume an additional fault at *Inv*3. Assuming these two faults is however not possible because the size of the diagnosis is restricted to 1 in this example. Therefore, DRUM–II terminates this branch of the search tree and backtracks.

```
|| Level: 7
|| Step:  [st_not(val(and1, i3, 1))]
|| Level: 8
|| Step:  [val(and1, i3, 0)]
|| Level: 9
|| Step:  [val(inv4, o, 0)]
|| Level: 10
|| Step:  [st_not(val(inv4, o, 1))]
...
|| Step:  [ab(nor1)]
 ... Solution (after 108 Flippings)
Saving Model 7
rev_state([7, 6, 5, 4, 3, 2], [val(inv1, i, 1),
val(inv2, i, 0), val(inv3, i, 0), diagnosis_phase(t),
val(nor1, o, 1)])
ab(nor1). with Rank: 1
ab(and1). with Rank: 1
ab(inv2). with Rank: 1
ab(inv4). with Rank: 1
ab(inv1). with Rank: 1
ab(inv3). with Rank: 1
```

After a total of 108 revision steps DRUM–II has identified all 6 single fault diagnoses for the given scenario. It saves the corresponding models (models 2 to 7) for further revision (output `rev_state` in the above trace).

4.1.4 Identifying Unnecessary Computations

The 9 bit parity checker shown in figure 4.3 is already relatively hard to solve for model–based diagnosers because of its reconvergent fanout: Every component influences most of the values in the circuit, so that there is a large number of different

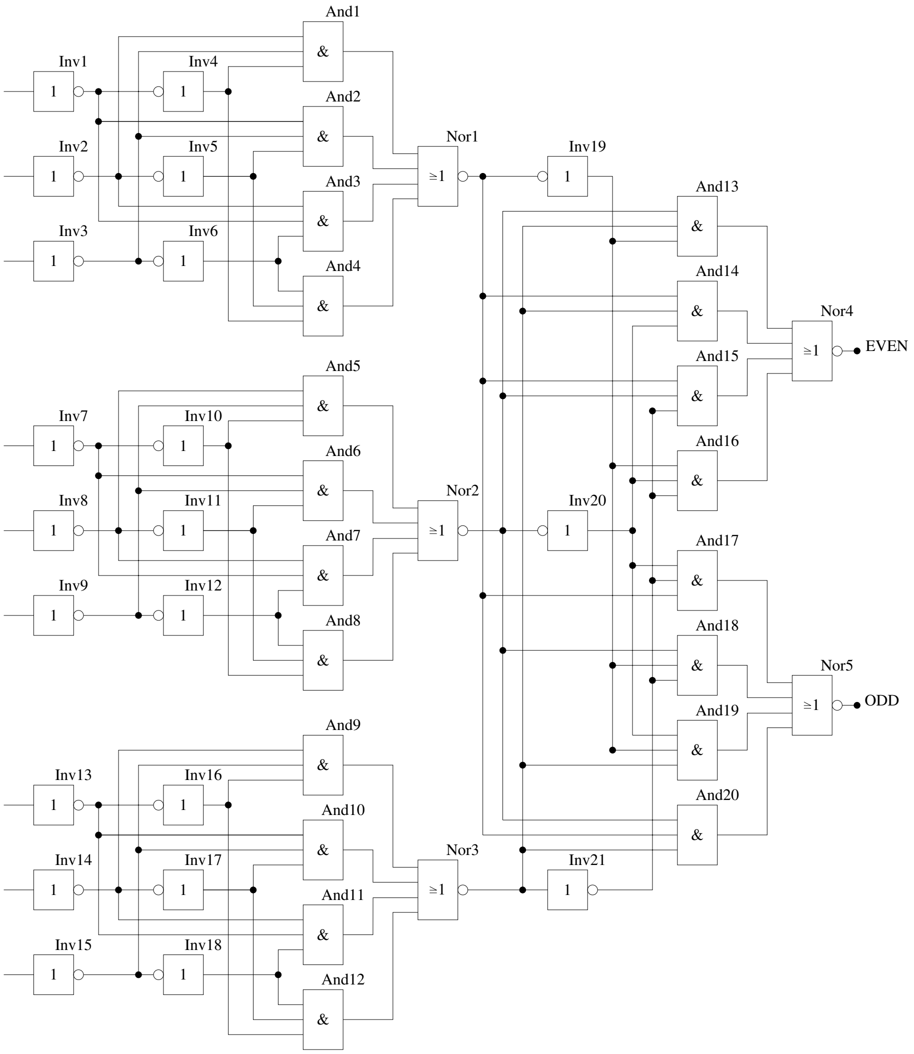


Figure 4.3: A 9-bit Parity Checker

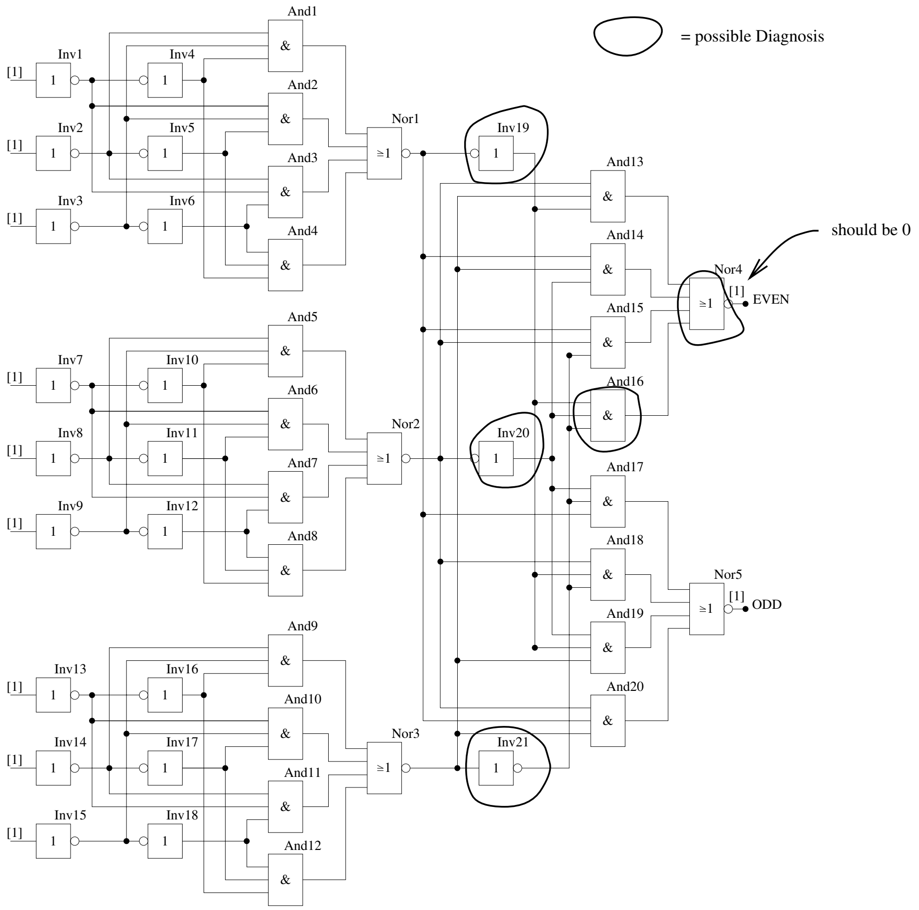


Figure 4.4: A Scenario together with all Single Fault Diagnoses

propagation paths which have to be considered. We will show why the performance of DRUM–II degrades quickly, when we move to the 9 bit circuit. Having identified the unnecessary computation carried out by the basic DRUM–II algorithm we will present an optimized version which solves large circuits very efficiently.

Let us point out that due to the modularity of model–based diagnosis, the system description of the 9 bit parity checker contains exactly the same axioms as the one for the 3 bit parity checker shown above. Only the sets of facts for the predicates *Type* and *Conn* are adapted to the different topology.

In the scenario from figure 4.4, we have computed all diagnoses of the sizes 1 to 4 using DRUM–II. Table 4.1 summarizes the results. The first column shows the limit on the diagnosis size with which DRUM–II was started. The second column gives the total number of diagnoses of the size specified in column 1. Finally, the third column gives the number of revision steps (i.e. flippings) executed during the computation of diagnoses.

These running times of the basic DRUM–II algorithm are not yet sufficient to di-

Diagnosis Size	Number of Diagnoses	Steps
0	0	616
≤ 1	5	4955
≤ 2	53	14145
≤ 3	245	41527
≤ 4	757	69229

Table 4.1: Running times of DRUM–II on the 9 bit parity checker

agnose large circuits efficiently. For example, the basic DRUM–II algorithm needs 30,22s to find all single fault diagnosis for the c499 circuit, which consists of 202 components. An examination of the traces produced by DRUM–II (for the 9 bit parity checker) reveals that it follows some propagation paths which can obviously not lead to diagnoses, given the structure of the circuit. Consider the following fragment from the trace of DRUM–II with a maximal diagnosis size of 1.

```
Assuming Abs[ab(and15)]([]) after 1243 Flippings
|| Level: 45
|| Step:  [ab(and15)]
|| Level: 46
|| Step:  [st_not(val(and17, i3, 0))]
```

Regardless of what happened in the previous 1243 flippings, these flippings on level 45 and 46 can never lead to a single fault diagnosis. The reason lies in the structure of the circuit (compare with figure 4.4): The output of $And15$ is not connected (directly or via intermediate components) to any of the inputs of $And17$. Consequently, if we change the value of $And17$'s third input, this change cannot be explained by a fault of $And15$. To find a consistent model, an additional fault has to be assumed.

Therefore, it is correct to consider the step on level 46 as a contradiction and terminate this propagation path. This simple optimization speeds up DRUM–II dramatically on large circuits. In the next section, we will formalize the idea just presented and prove its correctness.

4.2 Exploiting Structural Independence

We will now develop a theory of dependencies between literals in sets of horn clauses and apply it to diagnosis. The results presented below are applicable to all system descriptions, where the correct model can be formalized by propositional horn clauses. We do not assume that the whole system description horn since some clauses are typically not horn, e.g. relations among fault models like $\forall c\ Ok(c) \leftrightarrow \neg Ab(c)$.

4.2.1 Independence of Literals

Let us consider a set of Horn Clauses $\mathcal{S}$. Let $\{a_1, \ldots, a_n\}$ be the set of atoms occurring in $\mathcal{S}$. We say that a_j *depends directly on* a_i, if there is a clause containing a_j and $\neg a_i$.

Definition 4.1 a_j depends directly on a_i, *denoted by* $a_i \rightarrow^1 a_j$, *iff there is a clause* $C \in \mathcal{S}$ *with* $\{a_j, \neg a_i\} \subseteq C$.

We define the transitive closure of the direct dependency relation $\rightarrow^1$ and call it $\rightarrow^+$ (depends on).

Definition 4.2 *The relation* $\rightarrow^+$ *is defined inductively by*

1. *If* $a_i \rightarrow^1 a_j$, *then* $a_i \rightarrow^+ a_j$.

2. *If* $a_i \rightarrow^1 a_k$ *and* $a_k \rightarrow^+ a_l$, *then* $a_i \rightarrow^+ a_l$.

We say that a_j depends on a_i, *iff* $a_i \rightarrow^+ a_j$.

We want to study the effect of certain atoms on the other atoms of the theory. In particular, we want to see which changes are caused by assuming different sets A, B of atoms. If we know that an atom a was entailed by $\mathcal{S} \cup A$ (i.e. $\mathcal{S} \cup A \models a$), we want to know if a is still entailed if we assume the set of atoms B instead.

Proposition 4.3 *Let A and B be sets of atoms such that* $\mathcal{S} \cup A \not\models \perp$ *and* $\mathcal{S} \cup B \not\models \perp$. *Let a be an atom. If* $\mathcal{S} \cup A \models a$ *and there is no* $b \in \mathit{Diff}(A, B)$ *such that* $b \rightarrow^+ a$, *then* $\mathcal{S} \cup B \models a$.

Proof Sketch: The *positive* literal a follows from $S \cup A$, iff it holds in the minimal model of $S \cup A$. This minimal model coincides with the minimal model of $\mathcal{S}^+ \cup A$ ($\mathcal{S}^+$ consists of those clauses of $\mathcal{S}$, which contain a positive literal). Thus, $\mathcal{S}^+ \cup A \models a$. Thus, there is a proof for a, only based on the clauses in S^+. If we delete all unnecessary steps from this proof, only those steps remain, which derive a itself or literals on which a depends. This proof remains possible, if we replace A by B since we assume that A and B do not disagree on atoms, on which a depends. Thus, $\mathcal{S}^+ \cup B \models a$ and, since $\mathcal{S} \cup B$ is consistent, we conclude $\mathcal{S} \cup B \models a$. Q.E.D.

The knowledge from the above proposition can be used to focus the search for diagnoses.

4.2.2 Application to Diagnosis

In diagnosis, the horn clauses $\mathcal{S}$ are given by the system description *SD*. The following proposition is a special case of proposition 4.3.

Proposition 4.4 *Let SD be a system description, $OBS = OBS_{in} \dot{\cup} OBS_{out}$ an observation, a an atom and Δ a candidate (a set of components considered faulty).*

If $SD \cup OBS_{in} \cup \{Ok(c) | c \in Comp\} \models a$ and for all $c \in \Delta : Ok(c) \not\to^+ a$, then $SD \cup \{Ok(c) | c \in COMP \backslash \Delta\} \cup OBS \models a$.

To proof this proposition, we must assume, that the observations OBS_{out} are really the outputs of the device, i.e. no further atom depends on the output observations. The above proposition allows us to delete any model, which contains changes not influenced by the current candidate.

Corollary 4.5 *Let M be a model and $a \in M$ an atom occurring negatively in $Comm(M^i)$. Let Δ be a candidate. If for all $c \in \Delta : Ok(c) \not\to^+ a$ then $M \not\models SD \cup OBS \cup \{Ok(c) | c \in COMP \backslash \Delta\}$ and thus model M can be deleted.*

The method described above enables DRUM–II to eliminate a large number of candidates, which do not influence all abnormal outputs of the device as well as a large number of models which contain changes not influenced by the current candidate. The relation $\to^+$ needs only to be computed once for each system description. We compute $\to^+$ when the system description changes and store it in a (bit-) matrix, which is loaded into memory by the DRUM-II engine. The space complexity as well as the time complexity for computing the matrix is quadratic in the number of clauses of the underlying system description. A few comparisons with ATMS technology are worthwhile. First, while an ATMS computes actual dependencies of propositions at runtime, our approach computes possible dependencies at compile time. Second, as these possible dependencies are correct for each truth value of the input propositions, they obviously cannot exploit specific values of input propositions. An ATMS-environment is therefore more specific than our dependency set, and can exploit specific input values of the circuit. On the other hand, it seems impossible to precompile ATMS-dependencies since they depend on the actual test vector used. Third, as ATMSs usually store environments as bit vectors, the storage requirement for one environment is the same as for one of our dependency sets. Fourth, while a proposition usually can be deduced in different ways, and the ATMS therefore has to store more than one environment for these propositions, we have exactly one dependency set for each proposition, which avoids any explosion of these dependency sets. An interesting issue for further exploration would be to use our possible dependencies in an ATMS instead of the specific justifications computed by the ATMS at run-time in order to speed up ATMS-based diagnosis systems.

4.3 Combinatorial Benchmark Circuits

The ISCAS–85 benchmark suite contains combinatorial circuits with 160 to 3512 components. In recent years it has been regarded as a challenge for diagnostic engines in several papers [dK91, RdKS93, NF96, WN97] due to the size of the problems and their inherent complexity. Before we discuss the performance of DRUM–II on these problems and compare it to previous results, we provide a characterization of the kind of complexity present in these problems and discuss why some of these circuits are hard to solve.

4.3.1 Why are these Problems so difficult?

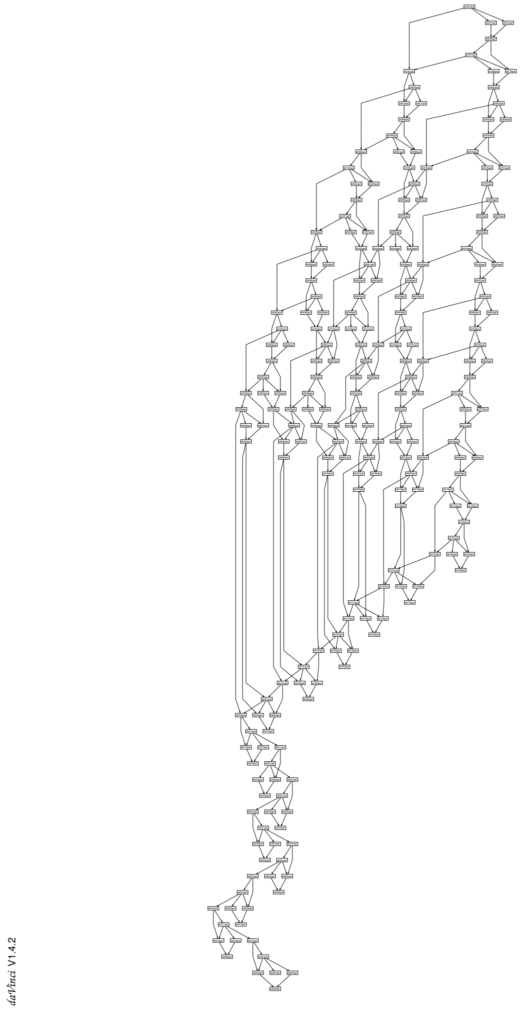


Figure 4.5: A Small Subgraph of the c6288 Circuit

For satisfiability and constraint satisfaction benchmarks consisting of random problems are widely used. Random problems are often characterized by a set of parameters $\langle n, m, p_1, p_2 \rangle$ [SD96], where n is the number of variables, m is the number of possible values per variable, p_1 is the constraint density (the probability that a constraint exists between two variables) and p_2 is the constraint tightness (the probability that two variables related by a constraint have incompatible values). The constraint tightness is not applicable to the circuits because the constraints (clauses) are not all binary. Other parameters for the ISCAS circuits are given in table 4.2. The estimated constraint density of the problems decreases when problem size increases[1]. This suggests that the problems are hard in a dimension not yet captured by random problems or at least by their usual numerical characterization. Figure 4.5 shows the graph structure of a small part of the c6288 circuit (all successors of a particular component). We can observe the phenomenon of reconvergent fanout, which was already noted by de Kleer [RdKS93]: Although a value occurs only in a small number of constraints, there are long chains of constraints, through which nearly all variables in the graph are finally related. For an algorithm this means that it is easy to find a locally consistent solution but after a long computation it will eventually turn out that the solution is not globally consistent. Consequently, the circuits are not easily solvable by brute force search alone and focusing is necessary. Moreover, there is a large number of long propagation paths making straightforward dependency recording algorithms run out of memory.

Circuit	Gates	n	m	p_1
c499	202	894	2	0.02
c880	383	1615	2	0.008
c1355	546	2238	2	0.007
c2670	1193	4928	2	0.002
c3540	1669	6377	2	0.002
c5315	2307	9356	2	0.002
c6288	2406	9696	2	0.002
c7552	3512	13582	2	0.001

Table 4.2: Statistical Parameters of the Circuits

4.3.2 Experimental Results

In this section we include experimental results for DRUM–II on the benchmark circuits. We have run 5 test vectors for each circuit. The running time is given in table 4.3. The times given are elapsed times in seconds on a lightly loaded SUN Ultra-1/170. They do not include model setup time, which is, however, very short (using a domain-specific forward chainer for generating the initial model).

Note, that the timing results show a large variation depending on the test vector used. This effect is striking with respect to the c6288. A reason for this is the different number of components which can explain the abnormal observation. Table 4.4 shows the number of propositions (possibly) influencing the abnormal observation for the

[1] Due to the special structure of the system description the actual constraint density is even lower than these estimates

Circuit	Running Time (sec)				
	t1	t2	t3	t4	t5
c499	0.013	0.036	0.040	0.036	0.013
c880	0.023	0.003	0.028	0.027	0.004
c1355	0.047	0.047	0.046	1.255	0.040
c2670	0.061	0.083	0.222	0.077	0.226
c3540	0.236	0.050	0.750	0.143	1.586
c5315	0.025	0.131	0.051	0.080	0.019
c6288	89.517	0.562	23.788	160.623	0.221
c7552	0.056	2.093	1.598	0.126	2.261

Table 4.3: Running Times of DRUM–II for 5 test vectors

different test vectors of the c6288. In the difficult cases t1 and t4 nearly the whole circuit influences the abnormal observation. We expect a large variation in running time for ATMS-based systems, too, though no such results have been published so far.

Circuit	t1	t2	t3	t4	t5
c6288	6316	795	2545	5804	375

Table 4.4: Propositions influencing the Abnormal Observation

Circuit	DRUM-II	de Kleer	IMPLODE	Williams
	Median	AAAI91	IJCAI–93	DX–96
c499	0.036	7.9	0.4	4.5
c880	0.023	6.2	0.8	4.0
c1355	0.047	242	1.4	12.3
c2670	0.083	33	3	28.8
c3540	0.236	1545	6	113.3
c5315	0.051	1215	7	61.2
c6288	23.788	–	8	–
c7552	1.598	1028	14	61.5

Table 4.5: Comparison of Running Times

In table 4.5 we compare the median running time of DRUM–II to previous results from the literature. DRUM–II is always the fastest system except for the c6288 where IMPLODE is faster. Since however the test vector used is not given in [RdKS93] it is not clear, whether IMPLODE's critical reasoning techniques provide still better focusing on the special structure of the c6288 for all test vectors.

4.4 Discussion

In this chapter we have discussed the modeling, simulation (i.e. model generation) and diagnosis of digital circuits with DRUM–II. We have shown that the efficiency of the DRUM–II algorithm can be increased dramatically by exploiting the structure of the device under consideration. While the unfocused DRUM–II needs 30 seconds for the diagnosis of a 200 gate circuit, the focused variant diagnoses circuits with up to 3500 components in a few seconds. On the circuits from the ISCAS–85 benchmark suite, which have a reconvergent structure and are therefore hard to solve for model–based reasoners, DRUM–II is more efficient than all systems reported in the literature.

Chapter 5

Model–Based Alarm Correlation with DRUM–II

In the diagnosis literature, it is often implicitly assumed that technical applications are best solved with consistency–based diagnosis, while abductive diagnosis is best suited for medical applications. In this chapter we discuss the application of DRUM–II to alarm correlation in cellular networks. It turns out that abduction is necessary to allow the use of an elegant model for this technical domain.

After giving an overview of the application domain, we will describe a consistency–based model implemented on top of DRUM–II. We assess the correctness of this model and the performance of its implementation using a library of representative test cases. Although the consistency–based model shows good performance, it is unnecessarily complex and only correct for networks with tree topology. We show that these drawbacks are due to weaknesses of the consistency–based approach to diagnosis. Then, we introduce an improved model using spectrum diagnosis, which is more intuitive and suitable for redundant topologies. We show that this model is efficiently evaluated by the spectrum diagnosis algorithm in DRUM–II. Preliminary results on the topics of this chapter have been described in [FNJW97]. In [WTJ$^+$97] we discuss alternative solutions to the alarm correlation problem.

5.1 Introduction

The growth of current cellular phone networks leads to an increase of administration costs within the terrestrial network required for relaying data and voice information. While most components within this net are semi-intelligent and send alarm messages when errors occur, the abstraction level of these alarm messages is usually too low, leading to a great number of alarms for any single cause. To avoid overloading the operators of these networks, alarm correlation systems are required which filter and condense the incoming alarms to meaningful high-level alarms and diagnoses. To mirror the dynamic nature of cellular phone networks, such an alarm correlation system should be easily reconfigurable for different topologies and extensions of network structure as well as for new and additional network elements. Moreover, missing alarms have to be tolerated without affecting the working of the system.

We give an overview over the problem and over previous solutions, and then describe an alarm correlation system based on a model–based approach. Our model describes the alarm behavior of the network elements, alarm propagation over these elements and the topology of the system. Reconfiguration of the network requires either simple updating of topology information or inclusion of additional components. Knowledge about new types of network elements can be easily integrated into the knowledge base.

5.2 Application Area

Mobile networks are growing rapidly. Satellite networks are used to reach subscribers around the world and are mainly used for locations where radio or cable coverage is difficult to achieve. Mobile radio networks were designed to interconnect land based vehicles to the terrestrial telephone networks. These networks mainly provide outdoor coverage, like the paneuropean digital GSM networks. Indoor mobility is achieved by extensions of cordless telephony using the digital DECT technology.

Alarm handling systems enable the network operator to run the network with minimal operating costs. The goal is to collect and interpret alarm messages and failure indications from the network elements without human intervention. The network has to adapt to most failures without additional user influence and support repair actions by supplying the relevant data to the technicians.

In large networks, like the current GSM networks, the alarm vectors supplied by the network elements tend to flood the workstations of the operators especially in critical situations like the passage of a thunderstorm front. Heavy rain affects the operation of microwave links and the electro magnetic power of lightning activates the protection switches used to safeguard the electronic equipment. Performance of the mobile network is degraded heavily in such situations and operators have difficulties interpreting the shower of important and less important messages from the network.

Mobile networks can be divided into three parts (see fig.5.1): the mobile station (MS), the access network with the base station transceivers (BTS) consisting of antennas, radio transceivers, cross connect systems (CC) and microwave (ML) or cable links (CL) and the base station controller (BSC), and the switched network, which is connected to the access network by the BSCs. The BSC provides the radio resource management, which serves the control and selection of appropriate radio channels to interconnect the MS and the switched network. The switched network interconnects the MS to the communication partner, which might be another MS or an ISDN subscriber.

Fig.5.2 show the structure of the alarm paths from the network elements (BTS, ML, CL, BSC) to the operation support system (OSS). The OSS hosts the operators

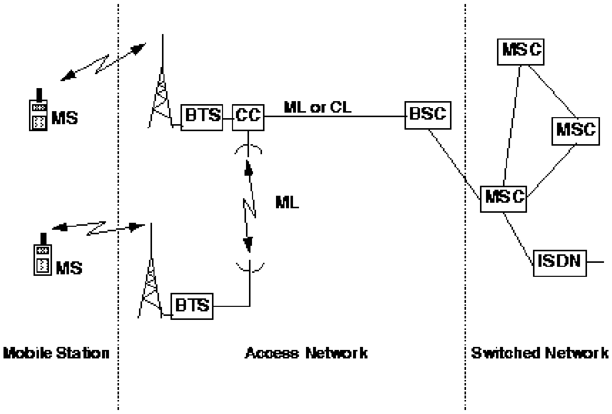


Figure 5.1: Structure of the GSM network

and performs all data processing necessary to filter, condense and interpret the alarm messages. The logical structure of the alarm network is shown in fig.5.2. The physical structure is different, but not relevant for our purpose. The alarms can be transmitted via the transmission network, the microwave or cable links, or via a specialized network, the X.25 network. The X.25 network is physically separated from the transmission network. The advantage of this separation is that a failure in the mobile network does not influence the alarm propagation. The disadvantage is the additional cost. Especially in the network elements located downwards from the BSC, communication and message paths are often combined to save costs.

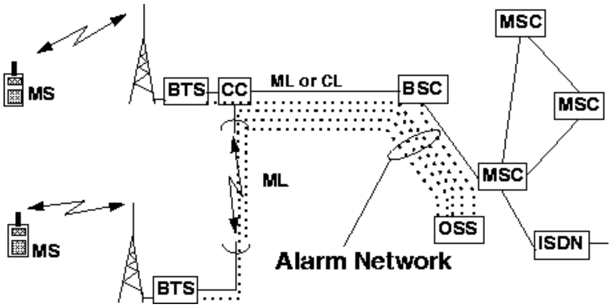


Figure 5.2: Alarm Network

A mediation device, which is located either in the network element, in the BSC or the OSS adapts the proprietary interfaces of the different vendors to the standard interfaces of the OSS. This mediation device can perform filtering and condensation functions of incoming messages. As the exact specification of this function is usually not known, the interpretation of alarm messages is further complicated.

5.3 Problem and Previous Solutions

5.3.1 Generation of Alarms

Due to fast and cost-efficient installation of links in base station subsystems, new operating companies realize most of their connections in the Base Station Subsystem (BSS) with microwave links. Other links are established with leased lines. The resulting network topology is a tree structure, where the traffic to several base station transceivers is distributed over a chain of microwaves and leased lines. Figure 5.3 shows an example. Cross–Connects are not displayed in figure 5.3 because we treat them, once initialized, as fixed and transparent connections. There is only one transmission path from a BTS to the related BSC.

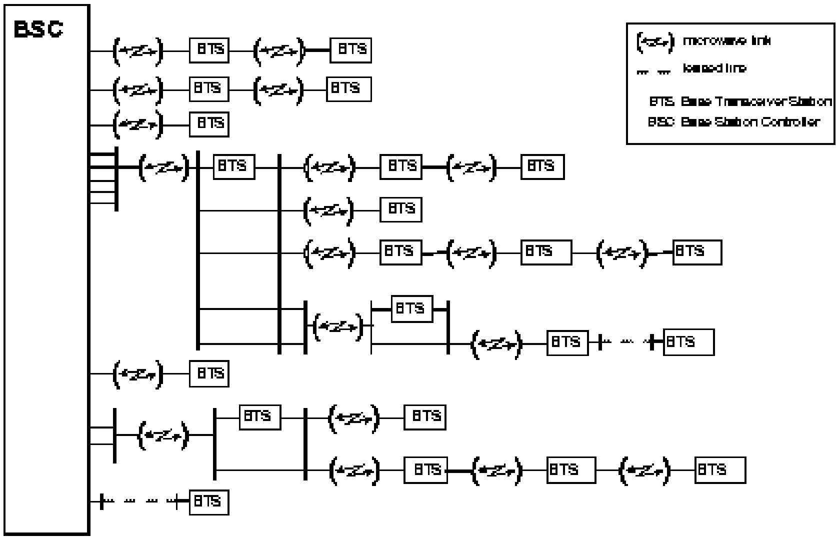


Figure 5.3: Configuration of a Base Station Subsystem

Alarm messages of BSC and BTS indicating link faults can be classified into three groups:

PCM-alarms: Alarms announcing problems with transmission on links. These alarms belong to failures in the physical layer of the Open System Interconnection (OSI) reference model (LOSS_OF_SIGNAL, AIS_RECEIVED, LOSS_OF_FRAME_ALIGNMENT, BIT_ERROR_RATE_>1E-3, FAREND_ALARM_1, PCM_FAILURE etc.)

Radio Resource alarms: Messages about failures in the management level of the radio resource ((Broadcast Channel) BCCH_MISSING, AVAILABLE_TRAFFIC (channel ratio below threshold)).

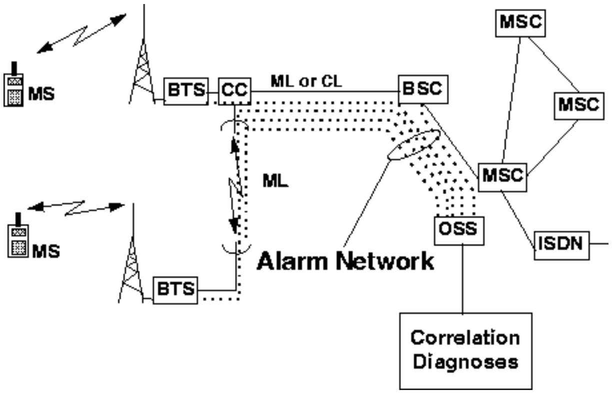


Figure 5.4: Correlation tool in the OSS

Alarms belonging to signaling- and O&M-connections: OSI-layer two and upper layers (FAILURE_IN_D-CHANNEL_ACTIVATION, LAPD_LINK_FAILURE, BTS_OMU_LINK_FAILURE).

Performance of microwave systems is weather dependent. Dense fog, heavy rain or snow can increase the bit error rate resulting in a connection breakdown. Such a breakdown of a physical connection interrupts voice connections as well as connections of control messages. As a consequence, up to 100 alarms are generated and transmitted to the OMC for a single failure. The operators in the OSS face several problems. First, a lot of alarms are forwarded to the OSS and have to be handled by the staff. More important alarms have to be separated from less important ones. The identification of the original failure is necessary as no alarms are available to directly indicate this failure. Operators are under stress, and reaction time to faults increases. Important alarms are misinterpreted or overlooked. Observations in the OSS have shown that alarm patterns belonging to the same fault do not match exactly as the original alarm patterns are disturbed by noise. This noise can result from other faulty or fluttering devices or delays in the transmission of alarms to the OSS. Prefiltering mechanisms in mediation devices and overload (e.g. in the transmitting system or mediation device) can also generate noise. A tool supporting the operating staff in the task of alarm- and fault-management is necessary to speed up reaction time to faults. This tool has to condense alarms, correlate them and precisely diagnose initial causes to achieve better quality of service. It is connected to the network management platform in the OSS, as show in Fig. 5.4.

5.3.2 Previous Solutions

A Coding Approach

In the coding approach [Yea96] each link-failure in the managed network is represented by an alarm vector. The binary alarm vector contains 1's for generated alarms and 0's otherwise. The alarm vectors for all links are collected in a codebook. At runtime the actual alarm vector is compared to the vectors in the codebook by calculating the Hamming-distance. The fault with the smallest Hamming-distance to the current alarm vector is assumed to be the cause of the observed failure. The coding approach speeds up alarm correlation compared to the usual rule–based approaches. However, for large networks or multiple faults the codebooks produced can be huge and the codebooks have to be regenerated after each topology change.

Rule Based Alarm Correlation

Approaches to rule based alarm correlation are known from the literature. In [Bea93] an expert system for a transport network is shown. [MT95] describes an intelligent filter for a SDH-network. Problems of rule based approaches are the evaluation of rules and real time-diagnosis.

Alarm Correlation Using Element Hierarchies

The IMPACT system described in [JW93] uses a hierarchy of network element types as well as a network configuration model and message classes, which make it more configurable and modular than other previous systems. However, correlation of alarms is still done using heuristic rules, no explicit description of alarm behavior and alarm propagation is used.

A Model–Based Approach to Network Maintenance

The goal of the AIM system developed within the RACE project AIM [KNH91] is the maintenance of telecommunication networks, for example broadband ISDN networks. The system shows the typical characteristics of a model–based system, using an explicit model of the telecommunication network as well as a model of its behavior. It therefore exhibits the advantages of model–based systems discussed in the next section, including easy maintenance, reconfiguration and extension. Being developed six years ago, however, the techniques used (based on a simple ATMS system) do not scale up well for large networks. Also, the network diagnosed consists of rather unintelligent network elements, which do not have the ability to diagnose local faults and generate alarms for such faults.

The RACE 2 project [SPBL95, dS95] also applied model–based techniques to the diagnosis and maintenance of a telecommunication network. The goal of this project was to support the on–line maintenance of heterogeneous networks. In contrast to the current work, this project focused on telecommunication networks with cable links.

5.4 A Consistency–Based Model

5.4.1 Overview of the Necessary Model

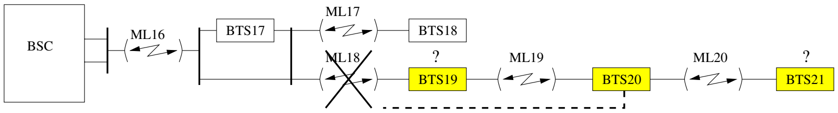
As discussed before, model–based diagnosis needs a model *SD* of the behavior of the system to simulate correct and/or faulty behavior and compare its results with the observations *OBS*. The set of components *COMP* considered interesting are the network elements which can be faulty. The observations we evaluate are alarm messages received / generated at the base station controller. We therefore have to model the alarm behavior as well as the propagation of alarms over network elements and connections between them to explain the existence or non-existence of alarm messages. This model has to be modular, i.e. basically describing behavior and propagation information for each type of network element. Additionally, topology information is included as a set of facts. In this way, new network elements can be easily added by just changing / extending the facts representing the topology information. New types of network elements (with possibly different alarm behavior) can be added by adding a description of the appropriate type together with a description of its alarm behavior.

We model base station transceivers, base station controllers and microwave links as basic network elements, which are connected by a tree–structured network. Data and alarm messages are sent through these elements and connections between them. In our current model, only microwave links may be assumed to be faulty, as this is the most interesting fault in these networks and is not handled by the current fault management software. A diagnosis therefore consists of one or more microwave links (considered faulty). Other faults, e.g. of base station transceivers can be (and actually are) handled by a simple alarm evaluation system, which can use a one-to-one correspondence between alarm messages generated for these faults and faulty network elements. The model–based alarm correlation system is used for those more complicated cases, when there is no one-to-one correspondence.

From the six alarm messages related to the base station controller and ten alarm messages related to the base transceiver station we only use five of the latter ones, as the alarms are quite redundant. Moreover, a detailed model for these alarms (involving the protocols on level 1, 2 and above) is not necessary. It is sufficient to divide these alarm messages into two classes, *farend* and *bts_failure* messages.

farend alarms are generated by a component if the components connected to this component on the down side (further away from the BSC) are not reachable any more. *bts_failure* messages for a component are generated if this component is not reachable from a BSC. *bts_failure* messages are generated directly in the BSC, when it detects (using the existence or non-existence of signals from the level 1 protocol) that the base station transceivers are not reachable. As an abstract model, we assume that periodical *alive* messages are sent by each component to the BSC. If the path from BSC to component is disrupted by a faulty component, these messages cannot be delivered to the BSC and the BSC generates the appropriate alarm message. Notice, that such a description of the alarm behavior does not necessitate a detailed description of the underlying protocols.

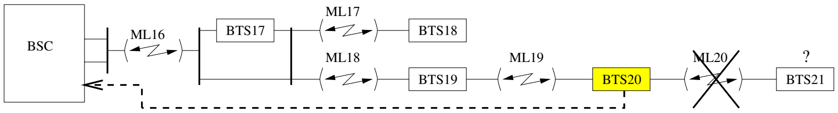
Propagation over connections and base station transceivers is trivial, as we assume these to be correct all the time. The difficult network elements are the microwave links. Let us look at the relevant two cases in detail using two examples. First, we get a *bts_failure* alarm for BTS20. This indicates that BTS20 is not reachable from the BSC. As poll messages can be lost only between the BSC and the BTS, and only a faulty microwave link can explain message loss, the three micro wave links between BSC and BTS20 are possible diagnoses. The following picture shows the model assuming the second microwave link to be faulty. In this case we predict poll messages for each BTS located downstream the faulty microwave link, i.e. BTS19, BTS20 and BTS21. We check the additional predictions for each diagnosis candidate to distinguish between the possible diagnoses.



The corresponding alarm rule can be phrased like this:

If we have a *bts_failure* alarm message for network element BTS_i
Then
the BSC has not received the *alive* message from component BTS_i

On the other hand, assume we get a *farend* alarm from *BTS20*. This *farend* alarm tells us, that the components located downstream of *BTS20* are not reachable. Using this observation, we can conclude that the faulty microwave link is located downstream of *BTS20*, in this case narrowing the set of diagnosis candidates to one.



The corresponding alarm rule can be phrased like this:

If we get an *farend* alarm from component BTS_i
Then
component BTS_i has sent a *farend* signal to the BSC
and this signal has not been discarded on the way to the BSC

5.4.2 Specific Model

The following set of formulas expresses these informal specifications. We use lowercase names for variables, uppercase names for constants and capitalized names for predicates.

First, we specify the signals used as well as their class (*BTS_FAILURE*) alarm or *FAREND* alarm. The *ALIVE* signal is not an alarm, but a status signal from the base station transceivers.

$$\begin{array}{l}
\mathit{Class}(\mathit{BTS\_OMU\_LINK\_FAIL}, \mathit{BTS\_FAILURE\_SIGNAL}).\\
\mathit{Class}(\mathit{BCCH\_MISSING}, \mathit{BTS\_FAILURE\_SIGNAL}).\\
\mathit{Class}(\mathit{AVAILABLE\_TRAFFIC}, \mathit{BTS\_FAILURE\_SIGNAL}).\\
\mathit{Class}(\mathit{LAPD\_LINK\_FAILURE}, \mathit{BTS\_FAILURE\_SIGNAL}).\\
\mathit{Class}(\mathit{FAREND\_ALARM\_1}, \mathit{FAREND\_SIGNAL}).\\
\mathit{Class}(\mathit{ALIVE}, \mathit{STATUS\_SIGNAL}).
\end{array}$$

Similarly, a set of facts describes the network elements and their types.

$$\begin{array}{l}
\mathit{Type}(\mathit{ML1}, \mathit{ML}).\mathit{Type}(\mathit{ML2}, \mathit{ML}).\mathit{Type}(\mathit{ML3}, \mathit{ML}).\\
\mathit{Type}(\mathit{ML4}, \mathit{ML}).\mathit{Type}(\mathit{ML5}, \mathit{ML}).\\
\ldots\\
\mathit{Type}(\mathit{BTS1}, \mathit{BTS}).\mathit{Type}(\mathit{BTS2}, \mathit{BTS}).\mathit{Type}(\mathit{BTS3}, \mathit{BTS}).\\
\mathit{Type}(\mathit{BTS4}, \mathit{BTS}).\mathit{Type}(\mathit{BTS5}, \mathit{BTS}).\\
\ldots
\end{array}$$

The topology is described by a set of connection facts. We denote the upstream port of each network element by *UP*, i.e. the port directed towards the *BSC*, and the opposite, downstream port by *DOWN*. For example, $\mathit{Conn}(ML16, UP, BSC, DOWN)$ means that the *UP* port of *ML16* is connected to the *DOWN* port of the *BSC*). When the topology of the network is changed, only this set of connection facts and the type facts described above have to be changed.

$$\begin{array}{ll}
\mathit{Conn}(\mathit{ML16}, \mathit{UP}, \mathit{BSC}, \mathit{DOWN}). & \mathit{Conn}(\mathit{BTS17}, \mathit{UP}, \mathit{ML16}, \mathit{DOWN}).\\
\mathit{Conn}(\mathit{ML18}, \mathit{UP}, \mathit{ML16}, \mathit{DOWN}). & \mathit{Conn}(\mathit{ML17}, \mathit{UP}, \mathit{BTS17}, \mathit{DOWN}).\\
\mathit{Conn}(\mathit{BTS18}, \mathit{UP}, \mathit{ML17}, \mathit{DOWN}).\ldots &
\end{array}$$

The following formulas describes the alarm behavior as well as the alarm propagation. First, we describe the abstraction from specific *bts_failure* alarms for a network element to an abstract observation, that (at least one) *bts_failure* alarm message has been received / generated for a specific network element. The predicate $Alarm(sender, PCM\_FAILURE)$ represents the observation, that the alarm message $PCM\_FAILURE$ has been received / generated for the network element *sender*, where *sender* can be any network element of type BTS.

$$\begin{array}{l}
\forall \mathit{sender}\\
\mathit{Type}(\mathit{sender}, \mathit{BTS}) \rightarrow\\
(\mathit{Bts\_Failure\_Alarm}(\mathit{sender}) \equiv\\
\;Alarm(\mathit{sender}, \mathit{BTS\_OMU\_LINK\_FAIL}) \vee\\
\;Alarm(\mathit{sender}, \mathit{BCCH\_MISSING}) \vee\\
\;Alarm(\mathit{sender}, \mathit{AVAILABLE\_TRAFFIC}) \vee\\
\;Alarm(\mathit{sender}, \mathit{LAPD\_LINK\_FAILURE}))
\end{array} \tag{5.1}$$

We assume that each base station transceiver sends an *alive* message and this message is present at its *UP*-port. The fact $Signal(ne1,UP,ne2,ALIVE)$ means, that the signal *ALIVE* sent from network element *ne2* is present at the port *UP* of network element *ne1*.

$$\begin{array}{l} \forall sender \\ Type(sender,BTS) \rightarrow Signal(sender,UP,sender,ALIVE) \end{array} \tag{5.2}$$

If we have observed a *bts_failure* alarm from a given network element, we can infer that no *alive* signal for this network element has been received at the base station controller *BSC*.

$$\begin{array}{l} \forall sender \\ Type(sender,BTS) \wedge Bts\_Failure\_Alarm(sender) \rightarrow \\ \neg Signal(BSC,DOWN,sender,ALIVE) \end{array} \tag{5.3}$$

If a *farend* alarm from a network element has been observed, then that element has sent this *farend* alarm and it has not been discarded on the way from the sender to the base station controller.

$$\begin{array}{l} \forall signal,sender \\ Class(signal,FAREND\_SIGNAL) \wedge Type(sender,BTS) \wedge \\ Alarm(sender,signal) \rightarrow \\ Signal(sender,UP,sender,signal) \wedge \neg Signal\_Discarded(sender) \end{array} \tag{5.4}$$

Signals are propagated over connections (into the direction of the base station controller).

$$\begin{array}{l} \forall signal,sender,ne1,ne2 \\ Type(sender,BTS) \wedge \\ (Class(signal,FAREND\_SIGNAL) \vee\ Class(signal,STATUS\_SIGNAL)) \wedge \\ Conn(ne1,UP,ne2,DOWN) \wedge \\ Signal(ne1,UP,sender,signal) \rightarrow \\ Signal(ne2,DOWN,sender,signal) \end{array} \tag{5.5}$$

Signals are also propagated over base stations (into the direction of the base station controller).

$$\begin{array}{l} \forall ne,sender \\ type(ne,BTS) \wedge type(sender,BTS) \wedge ne \neq sender \wedge \\ (Class(signal,FAREND\_SIGNAL) \vee Class(signal,STATUS\_SIGNAL)) \wedge \\ Signal(ne,DOWN,sender,signal) \rightarrow \\ Signal(ne,UP,sender,signal) \end{array} \tag{5.6}$$

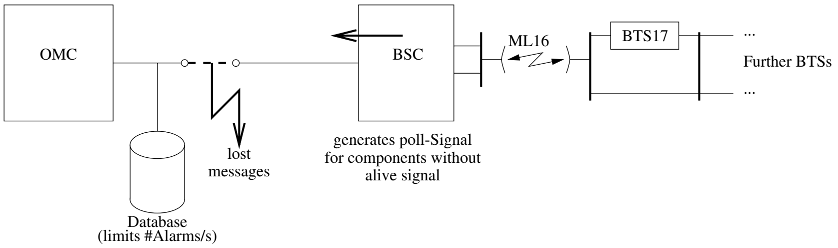


Figure 5.5: Transmission of alarms from BSC to OMC

Signals are also propagated over microwave links. If the microwave link is working correctly, it propagates a signal just like a connection or a base station transceiver. If the microwave link is defect, the signal is discarded.

$$\begin{array}{l}\forall ne, sender\\ Type(ne, ML) \wedge Type(sender, BTS) \wedge\\ (Class(signal, STATUS\_SIGNAL) \vee Class(signal, FAREND\_SIGNAL)) \wedge\\ \neg Ab(ne) \wedge Signal(ne, DOWN, sender, signal) \rightarrow\\ Signal(ne, UP, sender, signal)\end{array} \tag{5.7}$$

$$\begin{array}{l}\forall ne, sender\\ Type(ne, ML) \wedge Type(sender, BS) \wedge\\ (Class(signal, STATUS\_SIGNAL) \vee Class(signal, FAREND\_SIGNAL)) \wedge\\ Ab(ne) \wedge Signal(ne, DOWN, sender, signal) \rightarrow\\ Signal\_Discarded(sender)\end{array} \tag{5.8}$$

Finally, if a signal is discarded, a *bts_failure* alarm is generated.

$$\begin{array}{l}\forall sender\\ Type(sender, BS) \wedge Signal\_Discarded(sender) \rightarrow Bts\_Failure\_Alarm(sender)\end{array} \tag{5.9}$$

A further enhancement of this specification of alarm behavior concerns the predictions of each candidate model. In the model described so far all effects are deterministic. The model specifies for example, that we have to observe a *bts_failure* alarm for each component below a faulty microwave link. These predictions are not totally accurate, as filtering mechanisms within the network drop some alarm messages. Moreover, as shown in figure 5.5 some alarms are lost during alarm bursts due to the limited number of alarms, which the OMC database can record in a given time interval. We can easily extend our model to include the possibility of such lost alarms by assigning a probability to such events[1]. This allows us to tolerate lost alarms, as long as we

[1] An exact model of these processes would be very complex

have enough other messages indicating the faulty component. If too many alarms are dropped (meaning that we decrease the amount of evaluable alarm messages), more diagnosis candidates will be produced.

In our system description, we model the loss of an alarm message as a fault. The predicate $Ab(Transmission, Message)$ is true, if the alarm message *Message* is lost. We extend axiom 5.9 by the possibility that the alarm message is lost.

$$\begin{array}{l} \forall sender \\ Type(sender, BS) \wedge Signal\_Discarded(sender) \rightarrow \\ Bts\_Failure\_Alarm(sender) \vee Ab(Transmission, Bts\_Failure\_Alarm(sender)) \end{array} \tag{5.10}$$

5.4.3 Results

In this section we report the performance of our model–based approach on a library of 32 representative alarm cases.

The following cases represent a wide range of alarm patterns for the subnetwork shown in the previous examples (*BTS17* to *BTS21*). Table 5.1 is organized as follows: The first column shows the number of the test case, the second shows the reason for the alarm, i.e. the faulty microwave link. The third row shows the diagnosis proposed by DRUM–2, i.e. the maximally probable diagnosis. In row five the total a priori probability of the proposed diagnosis is shown. The next row contains the other minimal diagnoses, if present. Row 7 shows the relative probabilities of the most probable diagnosis compared to the runner up, e.g. 10:1 means that the proposed diagnosis is ten times as likely as the best competitor. Finally, the classification, i.e. correctness of the proposed diagnosis is denoted.

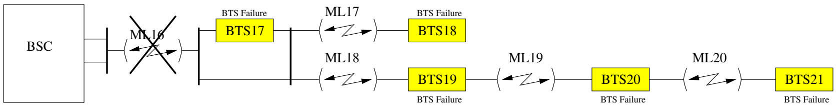
Due to the heavy traffic caused by defects of microwave links (alarm bursts), the probability of suppressed or lost *bts failure* alarms is rather high. For the test cases we assumed a probability of 0.1 of lost alarm messages and 0.01 for a faulty microwave link. The probabilities are much lower in reality, but for the purpose of diagnosis the exact values do not matter. The probabilities are needed only to discriminate between more plausible diagnoses (assuming less lost messages and faulty microwave links) and less plausible diagnoses.

In all test cases except one the system identifies the correct diagnosis, either as the single plausible diagnosis, or as the most probable diagnosis. We comment on the only exception in the next section.

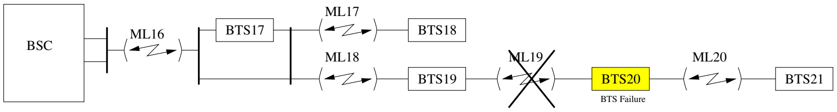
The running time of our prototype is also very encouraging. Table 5.2 shows the typical running time of our system for one test case. In the first row we show the time for the subnetwork used above. The second row shows the running time on the complete network of a large German city. All times were measured on a SUN Ultra 1 workstation.

5.4.4 Some Case Studies

Let us now examine three of our test cases in more detail to discuss the scope of our current approach. The first example obeys our first deterministic model as well as the probabilistic approach:



In this example microwave link $ML18$ is faulty and error messages are generated for all base transceiver stations located downstream. Since none of these messages is lost, the intended diagnosis $\{Ab(ML18)\}$ is found by both the deterministic and the probabilistic model. In the next example the message from *BTS21* is lost:



This case is still handled correctly by the probabilistic model, which yields the intended diagnosis

$$\Delta_1 = \{Ab(ML19), Ab(Transmission, Bts\_Failure\_Alarm(BTS21)\}$$

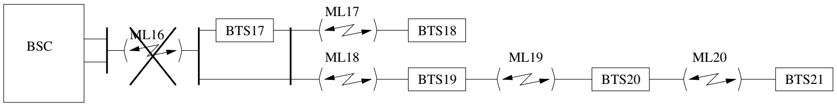
because the other minimal diagnoses

$$\begin{aligned}\Delta_2 \;\; = \;\; & \{Ab(ML18), Ab(Transmission, Bts\_Failure\_Alarm(BTS19)), \\ & Ab(Transmission, Bts\_Failure\_Alarm(BTS21))\}\end{aligned}$$

and

$$\begin{aligned}\Delta_3 \;\; = \;\; & \{Ab(ML1), Ab(Transmission, Bts\_Failure\_Alarm(BTS17)), \\ & Ab(Transmission, Bts\_Failure\_Alarm(BTS18)), \\ & Ab(Transmission, Bts\_Failure\_Alarm(BTS19)), \\ & Ab(Transmission, Bts\_Failure\_Alarm(BTS21))\}\end{aligned}$$

are less likely (since they assume more lost messages). Using the most probable diagnosis approach our system is able to handle 31 out of 32 alarm cases correctly. In the following case it produces no diagnosis, since all relevant alarm messages were lost or suppressed.



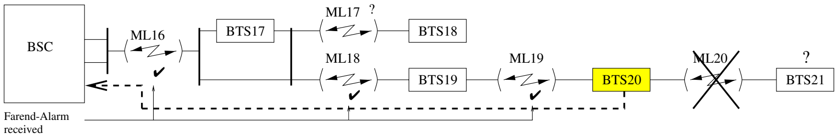
This exceptional case could be solved by integrating rules for additional *BSC*–specific signals into our model, which are however only present if the first BTS connected to the BSC is faulty.

5.5 An Improved System Description based on Spectrum Diagnoses

The consistency–based model discussed in the previous section has the advantage that it can be evaluated using every consistency-based diagnosis engine. However, by exploiting the expressiveness of DRUM–II, in particular our implementation of spectrum diagnoses (see section 3.6.2) we can define a model, which is both more general and more intuitive.

5.5.1 Limitations of the Consistency–Based Model

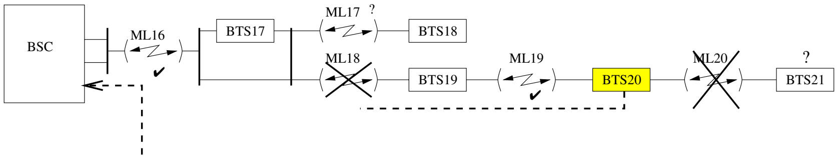
Before we introduce the improved model based on spectrum diagnoses, let us first point out the limitations of using purely consistency-based models. Consider the way we exploit farend–alarms: If a farend–alarm is received from a component, say $BTS20$, we know that all microwave links between $BTS20$ and the BSC must be working, because otherwise the signal would not have been transmitted.



Unfortunately, an intuitive knowledge base does not lead to this conclusion when consistency–based reasoning is applied. Suppose we only use the rule

$$\begin{aligned}
&\forall ne, sender\\
&Type(ne, ML) \wedge Type(sender, BTS) \wedge\\
&(Class(signal, STATUS\_SIGNAL) \vee Class(signal, FAREND\_SIGNAL)) \wedge\\
&\neg Ab(ne) \wedge Signal(ne, DOWN, sender, signal) \rightarrow\\
&Signal(ne, UP, sender, signal)
\end{aligned} \tag{5.11}$$

for describing the behavior of microwave links, which states that correctly functioning microwave links propagate signals. Then the following scenario would be consistent:



In this counter–intuitive scenario the farend–alarm is lost at the faulty microwave link $ML18$, but for some reason it reappears at the upstream port of the microwave link $ML16$. Obviously, this scenario cannot be avoided by adding a fault model for microwave links, because the alarm reappears at the upstream port of a correct microwave link. We could forbid such additional alarms by explicitly postulating that a signal can only appear at the upstream port of a microwave link, if it was present at one of its downstream ports:

$$\begin{array}{l} \forall ne1, sender, signal \\ Type(ne1, ML) \wedge Type(sender, BTS) \wedge Signal(ne1, UP, sender, signal) \rightarrow \\ (\exists ne2\ (Conn(ne2, UP, ne1, DOWN) \wedge Signal(ne2, UP, sender, signal))) \end{array} \tag{5.12}$$

But this artificial rule ruins the efficiency of the system. If a signal is observed somewhere in the network the system non–deterministically has to guess where it came from, to satisfy this rule.

In our system description from the previous section we avoided both the unintuitive scenarios and the computation overhead of the above rule by the *Signal_Discarded*–mechanism. However it is easy to see that this mechanism only works for a tree topology. Suppose there are two parallel transmission paths for a message. If one of these is not working due to a microwave link failure the system description predicts that the message is discarded. This is however wrong, since it can be transmitted over the correct parallel transmission path. When networks with redundant message paths are considered, a more general solution is needed.

The disadvantages of all consistency–based solutions to the interpretation of farend–alarms indicate, that consistency–based reasoning is too weak to solve this problem elegantly. On the contrary, if we postulate that received farend–alarms are explained abductively, we can use the simple rule 5.11 and no counter–intuitive scenarios will appear. The reason is, that the counter–intuitive scenarios are consistent with the alarm, but they do not explain how the alarm was transmitted.

5.5.2 System Description

We can use a very simple and intuitive model if we explain the farend–alarms abductively and use consistency–based reasoning for the bts failure alarms.

First we use the same rule as in the old model to denote that a *Bts_Failure_Alarm* is present if one of the poll–alarms has been observed.

$$\begin{array}{l} \forall sender \\ Type(sender, BTS) \rightarrow \\ (Bts\_Failure\_Alarm(sender) \equiv \\ \ Alarm(sender, BTS\_OMU\_LINK\_FAIL) \vee \\ \ Alarm(sender, BCCH\_MISSING) \vee \\ \ Alarm(sender, AVAILABLE\_TRAFFIC) \vee \\ \ Alarm(sender, LAPD\_LINK\_FAILURE)) \end{array} \tag{5.13}$$

Also, we assume that all components generate *ALIVE*–messages.

$$\begin{array}{l}\forall sender\\ Type(sender, BTS) \rightarrow Signal(sender, UP, sender, ALIVE)\end{array} \tag{5.14}$$

If the BSC receives an alive–message from a component, it does not generate an alarm for this component.

$$\begin{array}{l}\forall sender\\ Type(sender, BTS) \wedge Signal(BSC, DOWN, sender, ALIVE) \rightarrow\\ (\neg Bts\_Failure\_Alarm(sender) \wedge\\ \;\neg Ab(Transmission, Bts\_Failure\_Alarm(sender)))\end{array} \tag{5.15}$$

A bts failure alarm is displayed for a component *sender* if the *BSC* has not received an alive message from *sender* and if the bts failure alarm was not lost during transmission from BSC to OMC ($\neg Ab(Transmission, Bts\_Failure\_Alarm(sender))$).

$$\begin{array}{l}\forall sender\\ Type(sender, BTS) \wedge \neg Signal(BSC, DOWN, sender, ALIVE) \wedge\\ \neg Ab(Transmission, Bts\_Failure\_Alarm(sender)) \rightarrow\\ Bts\_Failure\_Alarm(sender)\end{array} \tag{5.16}$$

On the other hand, if the alarm message gets lost during transmission to the OMC, no alarm is generated.

$$\begin{array}{l}\forall sender\\ Type(sender, BTS) \wedge Ab(Transmission, Bts\_Failure\_Alarm(sender)) \rightarrow\\ \neg Bts\_Failure\_Alarm(sender)\end{array} \tag{5.17}$$

Farend alarms are displayed, if they are received by the BSC and they do not get lost during the transmission to the OMC.

$$\begin{array}{l}\forall sender\\ Signal(BSC, DOWN, sender, BCF\_BIE\_FAREND) \wedge\\ \neg Ab(Transmission, Far\_End\_Alarm(sender)) \rightarrow\\ Alarm(sender, BCF\_BIE\_FAREND)\end{array} \tag{5.18}$$

The generation of farend alarms is now very simple. A component generates a farend–alarm, if its downstream port is connected to a faulty microwave link.

$$\begin{array}{l}\forall ne1, ne2\\ Type(ne1, ML) \wedge Type(ne2, BTS) \wedge Conn(ne1, UP, ne2, DOWN) \wedge\\ Ab(ne1) \rightarrow Signal(ne2, UP, ne2, BCF\_BIE\_FAREND)\end{array} \tag{5.19}$$

Now we come to the propagation axioms. As in the consistency–based model, we assume that the connections (cable links) never fail. Thus, if a signal arrives at the downstream side of a connection it is propagated to the upstream side.

$$\begin{array}{l}
\forall \mathit{signal}, \mathit{sender}, \mathit{ne1}, \mathit{ne2} \\
\mathit{Type}(\mathit{sender}, \mathit{BTS}) \wedge \\
(\mathit{Class}(\mathit{signal}, \mathit{FAREND\_SIGNAL}) \vee \ \mathit{Class}(\mathit{signal}, \mathit{STATUS\_SIGNAL})) \wedge \\
\mathit{Conn}(\mathit{ne1}, \mathit{UP}, \mathit{ne2}, \mathit{DOWN}) \wedge \\
\mathit{Signal}(\mathit{ne1}, \mathit{UP}, \mathit{sender}, \mathit{signal}) \rightarrow \\
\mathit{Signal}(\mathit{ne2}, \mathit{DOWN}, \mathit{sender}, \mathit{signal})
\end{array} \tag{5.20}$$

Moreover, base transceiver stations propagate signals in the direction of the base station controller.

$$\begin{array}{l}
\forall \mathit{ne}, \mathit{sender} \\
\mathit{type}(\mathit{ne}, \mathit{BTS}) \wedge \mathit{type}(\mathit{sender}, \mathit{BTS}) \wedge \mathit{ne} \neq \mathit{sender} \wedge \\
(\mathit{Class}(\mathit{signal}, \mathit{FAREND\_SIGNAL}) \vee \mathit{Class}(\mathit{signal}, \mathit{STATUS\_SIGNAL})) \wedge \\
\mathit{Signal}(\mathit{ne}, \mathit{DOWN}, \mathit{sender}, \mathit{signal}) \rightarrow \\
\mathit{Signal}(\mathit{ne}, \mathit{UP}, \mathit{sender}, \mathit{signal})
\end{array} \tag{5.21}$$

Finally, signals are also propagated over microwave links. We only specify that a correctly functioning microwave link will forward a signal in the direction of the base station controller.

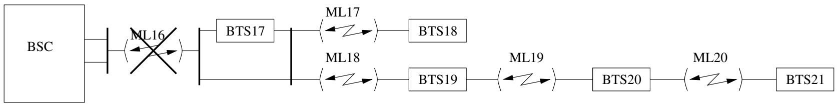
$$\begin{array}{l}
\forall \mathit{ne}, \mathit{sender} \\
\mathit{Type}(\mathit{ne}, \mathit{ML}) \wedge \mathit{Type}(\mathit{sender}, \mathit{BTS}) \wedge \\
(\mathit{Class}(\mathit{signal}, \mathit{STATUS\_SIGNAL}) \vee \mathit{Class}(\mathit{signal}, \mathit{FAREND\_SIGNAL})) \wedge \\
\neg \mathit{Ab}(\mathit{ne}) \wedge \mathit{Signal}(\mathit{ne}, \mathit{DOWN}, \mathit{sender}, \mathit{signal}) \rightarrow \\
\mathit{Signal}(\mathit{ne}, \mathit{UP}, \mathit{sender}, \mathit{signal})
\end{array} \tag{5.22}$$

Due to the greater discrimination power of spectrum diagnosis, no further information - like a fault model for microwave links - is needed.

5.5.3 Computing Diagnoses

Before computing spectrum diagnoses one has to decide, which part of the observations should be entailed logically. We want to logically entail the subset of the observations, which is due to messages arriving at the base station controller. Thus, we have to explain far end alarms, which are caused by farend messages arriving at the base station controller.

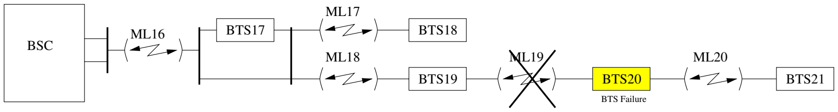
The reason why we want the system to explain farend messages abductively is that we cannot prevent the assumption of unnecessary alive messages by consistency–based reasoning (see section 5.5.1). For the same reason we want the system to explain the absence of bts failure messages, too.



In the above scenario we would expect bts failure alarms from all base transceiver stations. Suppose, the alarm from $BTS18$ got lost during transmission to the OMC. In our new system description, it is consistent that the alive message from $BTS18$ has arrived at the BSC although $ML16$ is not working and thus no alarm message was generated. However, if we force the system to explain the absence of the bts failure alarm, it can no longer assume that the alive message has arrived, because there is no causal explanation for its presence. Thus, the system must assume that the alive message did not arrive, and an alarm was generated in the BSC which was lost during the transmission to the OMC. This is our preferred explanation and we will see later in this section that it enables discrimination among several competing diagnoses. To summarize, we have:

$$\begin{aligned} Obs^{+} &= \{Alarm(c, BCF\_BIE\_FAREND) | \\ &\quad\; Alarm(c, BCF\_BIE\_FAREND) \in Obs\} \\ &\cup\; \{\neg Bts\_Failure\_Alarm(c) | \text{ no poll alarm for } c \text{ is contained in } Obs\} \end{aligned}$$

We will now show the steps executed during the computation of spectrum diagnoses using an example we already solved using the consistency–based model in section 5.4.4.



The observations in this example are

$$\begin{aligned} Obs \;&:=\; \{Alarm(BTS19, BCF\_BIE\_FAREND), \\ &\quad\; Alarm(BTS20, BCCH\_MISSING), \\ &\quad\; \neg Bts\_Failure\_Alarm(BTS17), \neg Bts\_Failure\_Alarm(BTS18), \\ &\quad\; \neg Bts\_Failure\_Alarm(BTS19), \neg Bts\_Failure\_Alarm(BTS20), \} \\ Obs^{+} \;&:=\; Obs \backslash \{Alarm(BTS20, BCCH\_MISSING)\} \end{aligned}$$

Let us now study the computation of spectrum diagnosis with algorithm 3.46. We start with a consistent model M of the network. In this model we have observed no alarms and all base transceiver station have sent alive messages, which have arrived at the base station controller. Now we revise this model with the observations Obs. We obtain three models:

$$
\begin{aligned}
M_1 \; := \; & \{Ab(ML16), Alarm(BTS20, BCCH\_MISSING), \\
& Alarm(BTS19, BCF\_BIE\_FAREND), \\
& Signal(BSC1, DOWN, BTS17, ALIVE), \\
& Signal(BSC1, DOWN, BTS18, ALIVE), \\
& Signal(BSC1, DOWN, BTS19, ALIVE), \\
& Signal(BSC1, DOWN, BTS21, ALIVE), \ldots\} \\
M_2 \; := \; & \{Ab(ML18), Alarm(BTS20, BCCH\_MISSING), \\
& Alarm(BTS19, BCF\_BIE\_FAREND), \\
& Signal(BSC1, DOWN, BTS17, ALIVE), \\
& Signal(BSC1, DOWN, BTS18, ALIVE), \\
& Signal(BSC1, DOWN, BTS19, ALIVE), \\
& Signal(BSC1, DOWN, BTS21, ALIVE), \ldots\} \\
M_3 \; := \; & \{Ab(ML19), Alarm(BTS20, BCCH\_MISSING), \\
& Alarm(BTS19, BCF\_BIE\_FAREND), \\
& Signal(BSC1, DOWN, BTS17, ALIVE), \\
& Signal(BSC1, DOWN, BTS18, ALIVE), \\
& Signal(BSC1, DOWN, BTS19, ALIVE), \\
& Signal(BSC1, DOWN, BTS21, ALIVE), \ldots\}
\end{aligned}
$$

All three models are eliminated in the following filtering step, because they all assume, that alive message have reached the base station controller although there was a faulty microwave link on the path, e.g. in all three models the alarm from $BTS21$ should not have reached the base station controller. Algorithm 3.46 now adds formulas to the system description, which rule out these three models.

$$
\begin{aligned}
SD' := \; & SD \\
\cup & \left\{ \bigvee_{\substack{Type(c,ML) \\ c \neq ML16}} Ab(c) \vee \bigvee_{\substack{Type(c,BTS) \\ Msg \in \{Bts\_Failure\_Alarm(c), \\ Far\_End\_Alarm(c)\}}} Ab(Transmission, Msg) \right\} \\
\cup & \left\{ \bigvee_{\substack{Type(c,ML) \\ c \neq ML18}} Ab(c) \vee \bigvee_{\substack{Type(c,BTS) \\ Msg \in \{Bts\_Failure\_Alarm(c), \\ Far\_End\_Alarm(c)\}}} Ab(Transmission, Msg) \right\} \\
\cup & \left\{ \bigvee_{\substack{Type(c,ML) \\ c \neq ML19}} Ab(c) \vee \bigvee_{\substack{Type(c,BTS) \\ Msg \in \{Bts\_Failure\_Alarm(c), \\ Far\_End\_Alarm(c)\}}} Ab(Transmission, Msg) \right\}
\end{aligned}
$$

Note, that lost alarm messages also count as failures, thus they are taken into account in the above addition of conflicts to the system description. The next consistency based diagnosis step with the new system description yields 16 diagnoses. Of these 16 diagnoses only one remains after the subsequent filtering step:

$$\begin{aligned} M \;:=\; \{ & Ab(ML19), Alarm(BTS20, BCCH\_MISSING), \\ & Alarm(BTS19, BCF\_BIE\_FAREND), \\ & Ab(Transmission, Bts\_Failure\_Alarm(BTS21)), \\ & Signal(BSC1, DOWN, BTS17, ALIVE), \\ & Signal(BSC1, DOWN, BTS18, ALIVE), \\ & Signal(BSC1, DOWN, BTS19, ALIVE), \ldots \} \end{aligned}$$

This model corresponds to the single most probable spectrum diagnosis. All alive messages at the base station controller are explained. The lost alive message from $BTS21$ is explained by the atom $Ab(Transmission, Bts\_Failure\_Alarm(BTS21))$. Although the system has to consider more diagnoses than in the consistency–based case the running time is still short, because the model is much simpler and thus each revision consists of very few steps. In the example above the overall running time for all diagnostic steps is 1.5s.

5.6 Discussion

In cellular phone networks faults of microwave links are likely to cause alarm showers, which put the system operator to a hard test. Alarm correlation tools are needed, which support the operators by identifying the cause of a large number of alarm messages.

In this chapter have presented the first fully model–based and scalable approach to alarm correlation, which is solely based on simulation of the alarm propagation behavior of the network. The main features of our solution are: (1) A very small and maintainable system description, that separates structural from behavioral components and thus makes changes of the network topology easy. (2) A propagation model which allows to correctly diagnose unforeseen errors as well as multiple faults. (3) Failure probability estimates, which lead to correct diagnoses even on noisy data, where alarm messages have been lost or suppressed.

In comparison to the circuit diagnosis problem we have studied in chapter 4 it is interesting to see that modeling the network is much more difficult and an elegant solution is only possible by the use of advanced diagnostic concepts. In our view this is due to the fact that the circuit description is fully deterministic while a natural description of the network describes only what happens to the different signals and does not provide a model of the microwave links as a function of all their parameters and inputs.

Due to its high accuracy and short running time our prototype is already suitable to assist the operator in real time.

No.	Fault	Diagnosis	a priori Probability	Competitors	Relative Probability	Class.
1	ML16	ML16	10^{-2}	–	n.a.	OK
2	ML16	ML16	10^{-2}	–	n.a.	OK
3	ML16	ML16	10^{-2}	–	n.a.	OK
4	ML16	ML16	10^{-3}	–	n.a.	OK
5	ML16	ML16	10^{-3}	–	n.a.	OK
6	ML16	ML16	10^{-4}	–	n.a.	OK
7	ML16	ML16	10^{-3}	–	n.a.	OK
8	ML16	–	–	–	–	BAD
9	ML17	ML17	10^{-2}	ML16	10000 : 1	OK
10	ML17	ML17	10^{-2}	ML16	10000 : 1	OK
11	ML17	ML17	10^{-2}	ML16	10000 : 1	OK
12	ML17	ML17	10^{-2}	ML16	10000 : 1	OK
13	ML17	ML17	10^{-2}	ML16	10000 : 1	OK
14	ML18	ML18	10^{-4}	ML16	100 : 1	OK
15	ML18	ML18	10^{-4}	ML16	100 : 1	OK
16	ML18	ML18	10^{-4}	ML16	100 : 1	OK
17	ML18	ML18	10^{-4}	ML16	100 : 1	OK
18	ML18	ML18	10^{-2}	ML16	100 : 1	OK
19	ML18	ML18	10^{-2}	ML16	100 : 1	OK
20	ML19	ML19	10^{-3}	ML18, ML16	10 : 1	OK
21	ML19	ML19	10^{-3}	–	n.a.	OK
22	ML19	ML19	10^{-3}	ML18, ML16	10 : 1	OK
23	ML19	ML19	10^{-3}	ML18, ML16	10 : 1	OK
24	ML19	ML19	10^{-3}	ML18, ML16	10 : 1	OK
25	ML19	ML19	10^{-2}	ML18, ML16	10 : 1	OK
26	ML19	ML19	10^{-2}	ML18, ML16	10 : 1	OK
27	ML19	ML19	10^{-2}	ML18, ML16	10 : 1	OK
28	ML20	ML20	10^{-3}	ML19, ML18	10 : 1	OK
29	ML20	ML20	10^{-3}	ML19, ML18	10 : 1	OK
30	ML20	ML20	10^{-3}	ML19, ML18	10 : 1	OK
31	ML20	ML20	10^{-3}	ML19, ML18	10 : 1	OK
32	ML20	ML20	10^{-3}	ML19, ML18	10 : 1	OK

Table 5.1: Results of our system on a set of test cases

Network	Number of BTSs	Number of MLs	Time
One Subnetwork	5	5	0.8s
A City Network	22	20	2.5s

Table 5.2: Running time of our system

Chapter 6

Tableaux for Diagnosis

The DRUM–II algorithm computes diagnoses by starting with a model of the correct behavior of an artifact and incrementally changing this model to reflect the symptoms observed. The use of a correct initial model is a key element of the DRUM–II algorithm because it focuses the search for diagnoses. In the following we will demonstrate that initial models can also be exploited using a standard theorem prover. The use of an initial model makes the tableau prover NIHIL capable of solving realistic diagnosis problems without further optimizations.

This chapter, which builds upon our previous work in [BFFN97a] and [BFFN97b], starts with a summary of the hyper tableau calculus, the formalism underlying NIHIL. Next, we discuss the benefits of using initial models. We will then define two approaches for integrating initial models into the hyper tableau calculus. Finally, we apply the focused theorem prover to the ISCAS–85 benchmark circuits.

6.1 Introduction

We will show how to integrate the idea to use initial models from our DRUM–II system into an implementation of the hyper tableaux calculus presented in [BFN96]. We refer to the result as *Semantic Hyper Tableaux*. The resulting system is comparable in efficiency to specialized systems for model–based diagnosis, although it currently lacks the ability of DRUM–II to exploit static dependencies. We know of no other general purpose theorem prover which has been used to solve large diagnosis problems.

The use of semantics within theorem proving procedures has been proposed before. There is the well-known concept of semantic resolution ([CL73]) and, more recently, there are approaches by Plaisted ([CP94] and Ganzinger ([GMW96]). Plaisted is arguing strongly for the need of giving semantic information for controlling the generation of clauses in his instance-based proof procedures, like hyper-linking. Ganzinger and his co-workers are presenting an approach where orderings are used to construct models of clause sets. Indeed, they even relate their approach to SATCHMO-like theorem proving, which is an instance of the hyper tableau calculus. However, the semantics has to be given by orderings or alternatively, by Horn subsets of the set of clauses. In cases where the initially given semantics is not compatible with orderings or is not expressible by Horn subsets it is unclear how to proceed. We will show, that in the case of diagnosis an initial semantics, which is naturally given by a model of the correct behavior of the device under consideration, can improve performance significantly. Our proof procedure does not impose any restrictions on these initial models.

6.2 Hyper Tableaux Calculus

In [BFN96] Baumgartner, Furbach and Niemelä introduced a variant of clausal normal form tableaux called "hyper tableaux". Hyper tableaux keep many desirable features of analytic tableaux (structure of proofs, reading off models in special cases) while taking advantage of central ideas from (positive) hyper resolution. We refer the reader to [BFN96] for a detailed discussion. In order to make the present chapter self-contained we will recall a simplified ground version of the calculus.

From now on $\mathcal{S}$ always denotes a finite ground clause set, and Σ denotes its signature, i.e. the set of all predicate symbols occurring in it. A *(clausal) tableau T* for a clause set $\mathcal{S}$ is an ordered tree t where the nodes are labeled with literals and in which, for every successor sequence $N_1, \ldots, N_n$ in t labeled with literals $K_1, \ldots, K_n$, respectively, there is a clause $\{K_1, \ldots, K_n\} \in \mathcal{S}$. In the following we will often identify nodes by their labels.

A *Branch* of a tableau T is a sequence $N_0, \ldots, N_n$ $(n \geq 0)$ of literals labeling nodes in T such that N_0 is the root of T, N_i is the immediate predecessor of N_{i+1} for $0 \leq i < n$, and N_n is a leaf of T. A branch $b = N_0, \ldots, N_n$ is called *regular* iff $N_i \neq N_j$ for $1 \leq i, j \leq n$ and $i \neq j$, otherwise it is called *irregular*. A tableau is *regular* iff every of its branches is regular, otherwise it is *irregular*. The set of *branch literals* of b is $lit(b) = \{N_1, \ldots, N_n\}$. We find it convenient to use a branch in place where a literal set is required, and mean its branch literals. For instance, we will write expressions like $A \in b$ instead of $A \in lit(b)$. In order to memorize the fact that a branch contains a contradiction, we allow to label a branch as either *open* or *closed*. A tableau is *closed* if each of its branches is closed, otherwise it is *open*. A *selection function* is a total function f which maps an open tableau to one of its open branches. If $f(T) = b$ we also say that *b is selected in T by f*. Fortunately, there is no restriction on which selection function to use. For instance, one can use a selection function which always selects the "leftmost" branch.

Definition 6.1 *Hyper tableau*
A literal set is called inconsistent *iff it contains a pair of complementary literals, otherwise it is called* consistent. Hyper tableaux *for a clause set $\mathcal{S}$ and a selection function f are inductively defined as follows:*

Initialization step: *The empty tree, consisting of the root node only, is a hyper tableau for $\mathcal{S}$. Its single branch is marked as "open".*

Hyper extension step: *If*

1. *T is an open hyper tableau for $\mathcal{S}$, $f(T) = b$ (i.e. b is the open branch selected in T by f), and*
2. *$C = A_1, \dots, A_m \leftarrow B_1, \dots, B_n$ is a clause from $\mathcal{S}$ ($m \geq 0$, $n \geq 0$), called* extending clause *in this context, and*
3. *$\{B_1, \dots, B_n\} \subseteq b$ (referred to as* hyper condition*)*

then the tree T' is a hyper tableau for $\mathcal{S}$, where T' is obtained from T by extension of b by *C: replace b in T by the* new *branches*

$$(b,A_1)\dots,(b,A_m),(b,\neg B_1)\dots,(b,\neg B_n)$$

and then mark every inconsistent new branch as "closed", and the other new branches as "open".

We will write the fact that T' can be obtained from T by a hyper extension in the way defined as $T \vdash_{b,C} T'$, and say that C is applicable *to b (or T).*

We say that a branch b is finished *iff it is either closed, or else whenever C is applicable to b, then extension of b by C yields some irregular new branch.*

The hyper condition of an extension expresses that *all* body literals have to be satisfied by the branch to be extended. This similarity to hyper *resolution* [Rob65] coined the name "hyper tableaux". From now on we only consider regular hyper tableaux. This restriction guarantees that for finite (ground) clause sets no branch can be extended infinitely often. This property is essential for the termination of the proof procedure.

Definition 6.2 *Branch Semantics*
For a set of propositions Σ we represent an interpretation I as the set $\{A \in \Sigma | I(A) = true\}$. Minimality *of interpretations is defined via set–inclusion.*

Let b be a consistent branch within a tableau. The branch b is mapped to the interpretation $[[b]]_\Sigma := lib(b)^+$, where $lit(b)^+ := \{A \in lit(b) | A \text{ is a positive literal}\}$. Usually we write $[[b]]$ instead of $[[b]]_\Sigma$ and let Σ be given by the context.

A *hyper tableaux derivation* is a sequence of tableaux, constructed by extension steps.

Definition 6.3 *Hyper Tableaux Derivation*
Let $\mathcal{S}$ be a finite clause set, called the set of input clauses*, and let f be a selection function. A (possible infinite) sequence $T_1, \dots, T_n, \dots$ of hyper tableaux for $\mathcal{S}$ is called a* hyper tableaux derivation from $\mathcal{S}$ *(or simply* derivation*) iff T_1 is obtained by an initialization step, and for $i > 1$, $T_{i-1} \vdash_{b_{i-1},C_{i-1}} T_i$ for some clause $C_{i-1} \in \mathcal{S}$. This is also written as $T_1 \vdash_{b_1,C_1} T_2 \cdots T_n \vdash_{b_n,C_n} T_{n+1} \cdots$. A hyper derivation is called* regular *iff every tableau in the derivation is regular, otherwise it is* irregular. *A hyper tableaux derivation is called a* hyper tableaux refutation *if it contains a closed tableau.*

A regular, finite hyper tableaux derivation from $\mathcal{S}$ of the given form is fair *iff it is a refutation or otherwise some open branch in the concluding tableau T_n is finished.*

The restriction to regular derivations is essential to guarantee that only finite derivations are possible. This holds immediately, because by König's Lemma infinite derivations are only possible if at least one branch is extended infinitely often. This however is impossible with a finite Σ (which we always assume finite, due to finite $\mathcal{S}$) and the restriction to have at most one occurrence of a literal in a branch.

For *fair* derivations, which end in a tableau with finished open branch b, the central property is that $[[b]]$ is a model for the given clause set $\mathcal{S}$. In other words, completeness holds. This is an instance of a more general result in [BFN96]. To apply hyper tableaux to the diagnosis task we additionally need the following completeness result.

Theorem 6.4 *Model Completeness of Hyper Tableaux.*
Let T be a hyper tableau for $\mathcal{S}$, such that every open branch is finished. Then, for every minimal model I of $\mathcal{S}$ there is an open branch b in T such that $I = [[b]]$.

Proof: If no minimal model for $\mathcal{S}$ exists, then the theorem trivially holds. Otherwise, let I be a minimal model for $\mathcal{S}$. In a first step we show that there is an open branch b such that $I \subset [[b]]$. It trivially holds that

$$\mathcal{S} \cup I \cup \bar{I} \text{ is satisfiable,} \tag{6.1}$$

where $\bar{I} := \Sigma \backslash I$, and for a set M: $\neg M := \{\neg A | A \in M\}$. Because of the minimality of I we can show that 6.1 is equivalent to $\mathcal{S} \cup \neg\bar{I} \models I$. This holds, iff

$$\mathcal{S} \cup \neg\bar{I} \cup \{\bigvee_{A \in I} \neg A\} \text{ is unsatisfiable.} \tag{6.2}$$

Hence, by refutational completeness of hyper tableaux there is a closed hyper tableau T' for this clause set. Further, by 6.1, the subset $\mathcal{S} \cup \neg\bar{I}$ is satisfiable. Hence, for the construction of T' the clause $\bigvee_{A \in I} \neg A$ must be used for an extension step, say at branch b. But, by definition of the hyper extension step this is possible only if the complementary literals are on branch b, i.e. $I \subseteq lit(b)^+$. We can omit from T' all extension steps with $\bigvee_{A \in I} \neg A$, as well as all extension steps with negative unit clauses $\neg\bar{I}$. The result is an open hyper tableau for $\mathcal{S}$ alone. Now, either the branch b is finished, and the theorem is proved, or otherwise T can be repeatedly extended so that at least one open finished branch b'' with $lit(b) \subseteq lit(b'')$ comes up. Reason: otherwise every such extension b'' of b would be closed, meaning that we could find a closed hyper tableau for $\mathcal{S} \cup \neg\bar{I}$ alone, which by soundness of hyper tableau contradicts the satisfiability of $\mathcal{S} \cup \neg\bar{I}$. Thus, b'' is the desired branch with $I \subseteq [[b'']]$. This concludes the first step of the proof.

Next, we show that for some branch b in T with $I \subseteq [[b]]$ we even have $I = [[b]]$. Suppose, to the contrary, for every branch b with $I \subseteq [[b]]$ we have $I \subset [[b]]$. That is, every such branch b contains a literal A with $A \notin I$. But then $A \in \bar{I}$. Hence b can be closed with $\neg A \in \neg\bar{I}$. If this is done for every such branch b, we can find a closed hyper tableaux for $\mathcal{S} \cup \neg\bar{I}$ alone, which, by soundness of hyper tableaux contradicts the satisfiability of $\mathcal{S} \cup \neg\bar{I}$. Hence, $I = [[b]]$ for some branch b in T. Q.E.D.

Before we discuss the integration of the initial model technique into hyper tableaux, let us study a derivation for a small circuit diagnosis problem.

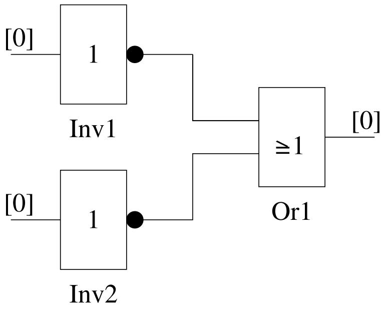


Figure 6.1: A Simple Fault Scenario

Example 6.5 *Hyper Tableau Derivation*
We will use the circuit in figure 6.1 as our running example. It can be described by the following clause set.

$$\begin{aligned}
Or1: \;& High(Or1,O) \rightarrow High(Or1,I1) \vee High(Or1,I2) \vee Ab(Or1)^1 \\
& High(Or1,I1) \rightarrow High(Or1,O1) \vee Ab(Or1) \\
& High(Or1,I2) \rightarrow High(Or1,O1) \vee Ab(Or1) \\
Inv1: \;& High(Inv1,I) \wedge High(Inv1,O) \rightarrow Ab(Inv1) \\
& High(Inv1,I) \vee High(Inv1,O) \vee Ab(Inv1) \\
Inv2: \;& High(Inv2,I) \wedge High(Inv2,O) \rightarrow Ab(Inv2) \\
& High(Inv2,I) \vee High(Inv2,O) \vee Ab(Inv2) \\
Conn1: \;& High(Inv1,O) \rightarrow High(Or1,I1) \\
& High(Or1,I1) \rightarrow High(Inv1,O) \\
Conn2: \;& High(Inv2,O) \rightarrow High(Or1,I2) \\
& High(Or1,I2) \rightarrow High(Inv2,O) \\
Obs_{In}: \;& High(Inv1,I) \\
& High(Inv2,I) \\
Obs_{Out}: \;& High(Or1,O) \\
& High(Inv1,I) \\
& High(Inv2,I)
\end{aligned}$$

Given the input observation Obs_{In} the above clause set predicts the following model for the correctly functioning circuit.

$$\begin{aligned}
I_0 = \{ \;& High(Inv1,O), High(Inv2,O), \\
& High(Or1,I1), High(Or1,I2), High(Or1,O)\}
\end{aligned}$$

This model will later serve as the initial interpretation. Figure 6.2 contains a hyper tableaux derivation.

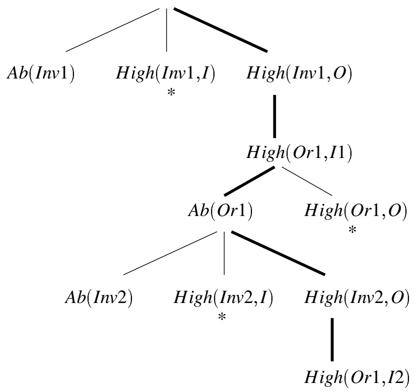


Figure 6.2: Hyper derivation.

In principle it is sufficient to compute models in order to solve the diagnosis task. In the derivation from figure 6.2 there are three open branches, each corresponding to a partial model. The rightmost branch is highlighted, because this model depicts an interesting aspect: remember, that a subset of the clauses stems from the description of the system under investigation. $SD := \{Or1, Inv1, Inv2, Conn1, Conn2\}$ is the system description. The highlighted model can be understood as an attempt to construct a model for the whole clause set, without assuming unnecessary *Ab*-predicates. Only for making the clauses from $Or1$ true it is necessary to include $Ab(Or1)$ into the model, because $High(Or1, O)$ cannot be assumed, since this contradicts the observation.

In order to guide the generation of models it is straightforward to use the knowledge of the clause set, i.e. to start with a model of the correct behavior of the device *SD* and to revise this assumption only if really necessary. This idea of a "semantically guided" model generation is a central idea of the DRUM–II system.

6.3 Lessons from DRUM–II

In order to make the generation of models efficient enough for problems of realistic size like the ISCAS–85 benchmark circuits, we exploit the "semantically guided" model generation of the DRUM-II system. As we already discussed in chapter 3 the basic idea of DRUM–II is to start with a model of the correct behavior of the device under consideration, i.e. with an interpretation I_0, such that $I_0 \models SD \cup \{\neg Ab(c) | c \in Comp\}$. Then the system description *SD* is augmented by an observation of abnormal behavior *Obs*, such that the assumption that all components are working correctly is no longer valid. Thus, I_0 is no model of $SD \cup Obs$, however it is used to guide the search for models of $SD \cup Obs$.

Consider the circuit from figure 6.3, which is an extension of our running example in figure 6.1. We observe that the output of the or–gate $Or1$ is wrong. The DRUM–II

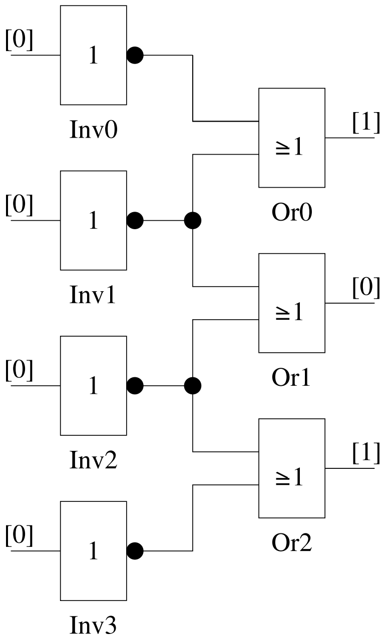


Figure 6.3: An example for the focusing effect of the initial model

algorithm starts diagnosis with a model of the correct behavior.

$$\begin{aligned} M := \quad & \{High(Inv0,O), High(Inv1,O), High(Inv2,O), High(Inv3,O), \\ & High(Or0,I1), High(Or0,I2), High(Or0,O), \\ & High(Or1,I1), High(Or1,I2), High(Or1,O), \\ & High(Or2,I1), High(Or2,I2), High(Or2,O)\} \end{aligned}$$

Then it inserts the output observations into this model. The symptom $\neg High(Or1,O)$, i.e. the low voltage observed at the output of the gate $Or1$ contradicts the correct behavior of the or–gate, in particular the following two clauses (compare with example 6.5):

$$High(Or1,I1) \rightarrow Ab(Or1) \vee High(Or1,O) \text{ and}$$
$$High(Or1,I2) \rightarrow Ab(Or1) \vee High(Or1,O)$$

Assuming $Ab(Or1)$ makes both clauses consistent again, yielding the diagnosis $\{Ab(Or1)\}$. The second possibility is to change the outputs of both inverters. Since the inputs of the inverters are observations, this can only be achieved by assuming that both inverters are abnormal. This yields the second minimal diagnosis $\{Ab(Inv1), Ab(Inv2)\}$.

DRUM–II has never considers any of the components *Inv*0, *Or*0, *Inv*3, or *Or*2 during the search for diagnoses. This is the effect of the initial model. Since the observations did not conflict with the values assumed for the components in the correct model, DRUM–II never considered changing these values.

Note, that the benefit gained by using an initial model is more than saving the computation of the expected output values of the additional components. The computation of the initial model is just one deterministic simulation of the circuit under the assumption that all components are working. It is obvious that an uninformed procedure would have to explore several useless alternatives during the search for models, i.e. assume that $Inv0$, $Or0$, $Inv3$, or $Or2$ are faulty. In fact, it has been shown in [NG94] that the use of an initial model leads to a constant diagnosis time for a sequence of n full adders, whereas the diagnosis time of uninformed algorithms is quadratic in n.

6.4 Formalizing the Diagnosis Task with Semantic Hyper Tableaux

In this section we discuss how to incorporate initial interpretations into the calculus.

Our first technique by *cuts* should be understood as the semantics of the approach; an efficient implementation by a *compilation technique* is presented afterwards.

6.4.1 Initial Interpretations via Cuts

The use of an initial interpretation can be approximated in the hyper tableau calculus by the introduction of an additional inference rule, the *atomic cut rule*.

Definition 6.6 *The inference rule* Atomic cut (with atom A) *is given by: if T is an open hyper tableau for $\mathcal{S}$, $f(T) = b$ (i.e. b is selected in T by f) where b is an open branch, then the literal tree T' is a hyper tableau for $\mathcal{S}$, where T' is obtained from T by extension of b by $A \vee \neg A$ (cf. Def. 6.1).*

Note that in regular tableaux it cannot occur that a cut with atom A is applied, if either A or $\neg A$ is contained on the branch. As a consequence it is impossible to use the "same cut" twice on a branch.

We approximate initial interpretations by applying atomic cuts at the beginning of each hyper tableau:

Definition 6.7 *An* initial tableau *for an interpretation I_0 is given by a regular tableau which is constructed by a applying atomic cuts with atoms from I_0 as long as possible.*

The branches of an initial tableau for an interpretation I_0 obviously consist of all interpretations with atoms from I_0. A part of the initial tableau for the initial interpretation I_0 given at the end of example 6.5 is depicted in Figure 6.4.1. Note that the highlighted branch corresponds to the highlighted part in Figure 6.2. If this branch is extended in successive hyper extension steps, the diagnosis $Ab(Or1)$, which was contained in the model from Figure 6.2 can be derived as well.

Note that the cuts introduce negative literals into a branch. The Definition 6.2 of the branch semantics applies to the calculus with cut as well: the interpretation associated

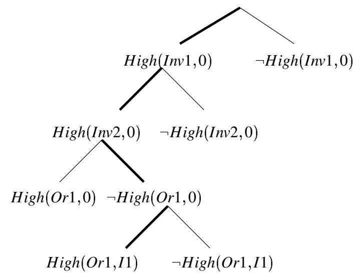


Figure 6.4: Initial tableau.

with a branch assigns *true* to an atom if it occurs positive on the branch, and a negated atom is interpreted as *false*, just as all atoms which are not on the branch.

In the following we take an initial tableau as the initialization step of a hyper tableaux construction. Since this initial tableau represents semantics, we call tableaux from such a derivation *semantic hyper tableaux*.

Definition 6.8 *Semantic Hyper Tableau – SHT*
A semantic hyper tableau *for I_0 and $\mathcal{S}$ is a hyper tableau which is generated according to Definition 6.1, except that the empty tableau in the initialization step is replaced by an initial tableau for I_0.*

It is easy to derive an open tableau starting from the initial tableau for I_0 in Figure 6.4.1, such that it contains the model from Figure 6.2. The model completeness of semantic hyper tableaux follows directly from theorem 6.4.

Theorem 6.9 *Model Completeness of SHT*
Let T be a semantic hyper tableaux for a set of clauses $\mathcal{S}$ and an interpretation I_0, such that every open branch is finished. Then, for every minimal model I of $\mathcal{S}$ there is an open branch b in T such that $I = [[b]]$.

6.4.2 Initial Interpretations via Renaming

The "semantical" account for initial interpretations defined in the previous section is not suitable for solving realistic examples. This is because all the $2^n \Leftrightarrow 1$ possible deviations from the initial interpretation will have to be investigated. In this section we introduce a compilation technique which implements the deviation from the initial interpretation only by need.

Assume we have an initial interpretation $I_0 = \{a\}$ and a clause set which contains $b \leftarrow$ and $c \leftarrow a \wedge b$. By the only applicable atomic cut we get the initial tableau with two branches, namely $\{a\}$ and $\{\neg a\}$. The first branch can be extended twice by an hyper extension step, yielding $\{a, b, c\}$. The second branch can be extended towards $\{\neg a, b\}$. No more extension step is applicable to this tableau. Let T_{cut} be this tableau.

Let us now transform the clause set with respect to I_0, such that every atom from I_0 occuring in a clause is shifted to the other side of the $\leftarrow$ symbol and complemented. In our example we get the clause $c \vee \neg a \leftarrow b$; the fact $b \leftarrow$ remains, because b is not in I_0. Using $b \leftarrow$ we construct a tableau consisting of the single branch $\{b\}$, which can be extended in an successive hyper extension step by using the renamed clause. We get a tableau consisting of two branches $\{b, c\}$ and $\{b, \neg a\}$. Let T be that tableau. Now, let us interpret a branch in T as usual, except that we set an atom from I_0 to *true* if its negation is not contained in the branch. Under this interpretation the branch $\{b, c\}$ in T corresponds to the usual interpretation of $\{a, b, c\}$ in T_{cut}. Likewise, the second branch $\{b, \neg a\}$ in T_{cut} corresponds to the second model in T.

Note that by this renaming we get tableaux where atoms from I_0 occur only *negatively* on open branches; such cases just mean deviations from I_o. In contrast to the cut approach, these deviations are now brought into the tableau by need.

The following definition introduces this idea formally. Since we want to avoid unnecessary changes to the hyper calculus, a new predicate name *neg_A* instead of $\neg A$ will be used.

Definition 6.10 *I-transformation*
*Let $C = L_1 \vee \ldots \vee L_n$ be a clause and I be a set of atoms. The I-*transformation of *C is the clause obtained from C by replacing every positive literal A with $A \in I$ by $\neg neg\_A$, and by replacing every negative literal $\neg A$ with $A \in I$ by neg_A. The I-*transformation of *$\mathcal{S}$, written as $\mathcal{S}^I$, is defined as the I-transformation of every clause in $\mathcal{S}$.*

It is easy to see that every I-transformation preserves the models of a clause set, in the sense that every model for the non-transformed clause set constitutes a model for the transformed clause set by setting *neg_A* to *true* iff A is *false*, for every $A \in I$, and keeping the truth values for atoms outside of I. More formally we have:

Proposition 6.11 *Model Preservation of I-transformation*
For every model J for $\mathcal{S}$: $J \models \mathcal{S}$ iff $rename_I(J) \models \mathcal{S}^I$, where $rename_I(J)(neg\_A) = \overline{J(A)}$ iff $A \in I$, and else $rename_I(J)(A) = J(A)$.

As explained informally above, the branch semantics of tableaux derived from a renamed, i.e. I-transformed clause set, is changed to assign *true* to every atom from I, unless its negation is on the branch. This is a formal definition:

$$\begin{aligned}[[b]]^I = \quad & (I \setminus \{A | neg\_A \in lit(b)\}) \cup \\ & (lit(b) \setminus \{neg\_A | neg\_A \in lit(b)\})\end{aligned}$$

The connection of semantic hyper tableaux to hyper tableaux and renaming is given by the next theorem.

Theorem 6.12 *Let T be a semantic hyper tableau for a clause set $\mathcal{S}$ and an initial model I where every open branch is finished; let T^I be a hyper tableau for the I-transformation of $\mathcal{S}$ where every open branch is finished. Then, for every open branch b^I in T^I there is an open branch b in T such $[[b^I]]^I = [[b]]$. The converse does not hold.*

The theorem tells us that with the renamed clause set we compute some deviation of the initial interpretation. The value of the theorem comes from the fact that the converse does not hold in general. That is, not every possible deviation is examined by naive enumeration of all combinations.

In order to see that the converse does not hold, take e.g. $\mathcal{S} = \{a \leftarrow\}$ and $I_0 = \{b\}$. There is only one semantic hyper tableau of the stated form, namely the one with the two branches $\{b, a\}$ and $\{\neg b, a\}$. On the other side, the I_0-transformation leaves $\mathcal{S}$ untouched, and thus the only hyper tableau for $\mathcal{S}$ consists of the single branch $\{a\}$ with semantics $[[\{a\}]]^{I_0} = \{a, b\}$. However, the semantics of the branch $\{\neg b, a\}$ in the former tableau is different.

6.5 Implementation and Experiments

We have implemented a proof procedure for the hyper tableaux calculus of [BFN96], modified it slightly for our diagnosis task, and applied it to some benchmark examples from the diagnosis literature.

The Basic Proof Procedure. A basic proof procedure for the plain hyper tableaux calculus for the propositional case is very simple, and coincides with e.g. SATCHMO [MB88]. Initially, let T be a tableau consisting of the root node only. In general, let T be the tableau constructed so far. *Main loop:* if T is closed, stop with “unsatisfiable”. Otherwise select an open branch b from T (branch selection) which is not labeled as “finished” and select a clause $H \leftarrow B$ (extension clause selection) from the input clause set such that $B \subseteq b$ (applicability) and $H \cap b \neq \{\}$ (regularity check). If no such clause exists, b is labeled as “finished” and $[[b]]$ is a model for the input clause set. In particular, the set of literals on b with predicate symbol Ab (simply called Ab-literals) constitutes a (not necessarily minimal) diagnosis. If every open branch is labeled as “finished” then stop, otherwise enter the main loop again.

In the diagnosis task it is often demanded to compute every (minimal) diagnosis. Hence the proof procedure does not stop after the first open branch is found, but only marks it as “finished” and enters the main loop again. Consequently, the “branch selection function” is not of real significance because every unfinished open branch will be selected eventually. However, the “extension clause selection” *is* an issue. A standard heuristic for tableau procedures is to select clauses first which avoid branching. For our diagnosis experiments, however, a clause selection function, which prefers clauses with some body literal being equal to the leaf of the branch to be extended, turned out to be superior.

For further improvements of the proof procedure, such as *factorization* and *level cut* see [BFN96].

Adaption for the Diagnosis Task. Recall that our diagnosis task requires to bias the proof search in two ways: incorporation of the initial interpretation, and implementing the n-faults assumption. While the former is treated by renaming predicates in the input clause set (Section 6.4), the latter is dealt with by the following new inference rule: "any branch containing $n+1$ (due to regularity necessarily pairwise different) Ab-literals is closed immediately". Notice that this inference rule has the same effect as if the $\binom{|Comp|}{n+1}$ clauses $\leftarrow ab(C_1), \ldots, ab(C_{n+1})$ (for $C_i \in Comp$, $C_i \neq C_j$, where $1 \leq i, j \leq n+1$ and $i \neq j$) specifying the n-faults assumption would be added to the input clause set. Since even for the smallest example (c499) and the 1-fault assumption the clause set would blow up from 1600 to 60000 clauses, the inference rule solution is mandatory.

n-fault Assumption and Computing Minimal Diagnoses. When computing diagnoses under the n-faults assumption the following *minimality properties* are evidently of interest: first, every diagnosis should be computed only once; second, every computed diagnosis should be minimal (Def. 2.1), i.e. if Δ is a diagnosis then no proper subset is a diagnosis.

We describe how both properties can be achieved by a combined iterative deepening/lemma technique, comparable to the iterative deepening algorithm 3.32. It can be implemented by a simple outer loop around the proof procedure. The outer loop includes a counter $N = 0, 1, 2, \ldots$, which stands for the cardinality of the diagnosis computed in the inner loop. The invariant for the outer loop is the following: *all minimal diagnoses with cardinality* $\leq N \Leftrightarrow 1$ *have been computed, say it is the set* Δ_{N-1}*, and every such diagnosis* $\{Ab(C_1), \ldots, Ab(C_n)\} \in \Delta_{N-1}$ *has been added to the input clause set as a* lemma clause $\neg Ab(C_1) \vee \ldots \vee \neg Ab(C_n)$. Before entering the proof procedure in the inner loop we set $\Delta_N := \Delta_{N-1}$, and the proof procedure is slightly modified according to the following rule: whenever a finished open branch b is derived, the Ab-literals on b are collected as new diagnosis Δ. Then we extend $\Delta_N := \Delta_N \cup \{\Delta\}$ and add Δ as a lemma clause $\bigvee_{L \in \Delta} \neg L$ to the input clause set. No more modifications are necessary.

Notice that the lemma clauses are purely negative clauses and hence can be used to close a branch. Since we give preference to negative clauses both minimality issues addressed above are accounted for, and hence the invariant follows. Reason: first, it is impossible to compute the same diagnosis more than once, because as soon as a diagnosis is computed the first time, it is turned into a (negative) lemma clause which will be used to immediately close all branches containing the same diagnosis. Second, we compute only minimal diagnoses as an immediate consequence of the iterative deepening over N: any branch containing a non-minimal diagnosis would have been closed by a lemma clause stemming from a diagnosis with strictly smaller cardinality, which must be contained in the input clause set due to the invariant for some value $< N$.

Although this procedure for computing minimal diagnoses under the n-fault assumption is simple, it has some nice properties. First, it can be implemented easily by slight modifications to the basic hyper tableau proof procedure. Second, in the hyper

Name	*# Gates*	*# Clauses*	*# Diagnosis*	*Time*	*# Steps*
C499	202	1685	2	5	3015
C880	383	2776	19	2	161
C1355	546	3839	5	47	24699
C2670	1193	8260	31	6	533
C3540	1669	10658	3	10853	1473572
C5315	2307	16122	5	13	3071

Figure 6.5: ISCAS'85 Circuits and NIHIL results.

tableau proof procedure we look at one branch at a time, which gives a polynomial upper bound for the memory requirements of the tableau currently constructed. Admittedly, there are possibly exponentially many minimal diagnosis wrt. $|Comp|$, but we assume that those will be kept anyway. Further, it is important to notice that we compute minimal models only wrt. the extension of the *Ab*-predicate, but not of minimal models wrt. all predicates. Since $|Comp|$ is usually much smaller than the number of atoms in the translation $(SD, Comp, Obs)$ to propositional logic, computing minimal models wrt. all predicates would usually require considerably more memory (such an approach to minimal model reasoning was proposed in [BY96]).

Implementation. Our prover is called *NIHIL* (*N*ew *I*mplementation of *H*yper *i*n *L*isp) and is a re-implementation in SCHEME of a former Prolog implementation. Because the SCHEME code is compiled to C, the basic performance is quite good. Obviously, since NIHIL is a first-order prover and the data structures are arranged for this case, there is a significant overhead when dealing with propositional formulas.

Experiments. For our experiments we ran parts of the ISCAS-85 benchmarks [Isc85] from the diagnosis literature. This benchmark suite consists of combinatorial circuits from 160 to 3512 components. Table 6.5 describes the characteristics of the circuits we used. NIHIL was set up as described above. The abovementioned optimizations *factorization* and *level cut* were tried, but had no influence on these examples.

The results are summarized in Table 6.5. *# Clauses* is the number of input clauses stemming from the problem description. *Time* denotes proof time proper in seconds, and thus excludes time for reading the problem and setup (which is less than about 10 seconds in any case). The times were taken on a SparcStation 20. *# Steps* denotes the number of hyper extension steps to obtain the final tableau, and *# Diag* denotes the number of single fault diagnoses. We emphasize that the results refer to the clause sets with renamed predicates according to Section 6.4. Without renaming, and thus taking advantage of the initial interpretation, only the c499 was solvable (in 174 seconds); all other examples could not be solved within 2 hours, whatever flag settings/heuristic we tried.

6.6 Discussion

In this chapter we analyzed the relationship between logic-based diagnostic reasoning and tableaux based theorem proving. We showed how to implement diagnostic reasoning efficiently using a hyper tableaux based theorem prover. We identified the use of an initial model as one of the central techniques in the diagnostic reasoning engine DRUM–II and showed how to apply this technique within a hyper tableaux calculus. The resulting theorem prover NIHIL is capable of diagnosing large benchmark circuits from the diagnosis literature. There are some open theoretical questions, e.g. we have to prove formally that by renaming we again get a model complete calculus and it would be interesting to characterize the computed models more exactly.

Chapter 7

Strategies for Diagnosis

In the previous chapters we have introduced an efficient diagnostic engine and evaluated it in several applications. The current chapter, which is based on our previous results described in [FNS94, NFS95, FNS, FNS96, FNS98] is dedicated to the process aspect of diagnosis. In practice, diagnosis is a dynamic process consisting of several diagnosis steps under changing assumptions; actions have to be performed during this process. Usually a symptom alone will not lead to a unique diagnosis. Measurements are needed to discriminate among competing hypotheses. Furthermore, there are often different system descriptions for one device. For example, electronic devices are modeled at several levels of detail. Each abstraction level can be regarded from the structural, physical or functional viewpoint.

In previous approaches to process support for model–based diagnosis different strategies and assumptions for this diagnostic process were encoded in the implementation of the diagnostic system. This solution is however not very flexible. Furthermore, it contradicts the separation of knowledge base and problem solving component. In this chapter we present a language for defining the diagnostic process as declaratively as the system description itself. We provide algorithms for efficiently exploiting these process descriptions and give guidelines for designing diagnostic processes.

7.1 Introduction

For the diagnosis of complex systems it is not sufficient to have only one static system model for which the minimal diagnoses are computed. In order to handle the complexity of diagnosis we have to see diagnosis as a dynamic process which is controlled by diagnosis assumptions. Struss [Str92] gave the first formalization of diagnosis as a process and made the diagnosis assumptions explicit by introducing the concept of a working hypothesis.

In the same spirit Boettcher and Dressler ([BD93], [BD94]) developed a catalogue of diagnosis strategies. Furthermore they provided an intuitive semantics and an ATMS–based implementation for these strategies. The disadvantage of their approach is that they use a static set of strategies which is coded into the diagnosis algorithm. Missing a declarative semantics for these diagnosis strategies independent of a particular implementation makes the definition of new or application–specific strategies more difficult than it should be.

In this chapter we extend their approach by introducing a formal meta–language for the definition of diagnosis strategies. This language makes strategies explicit and allows to define strategies specific to an application similar to defining system models. So our framework extends model–based diagnosis in the sense that not only the behavior of the system but also the strategic knowledge about the system model is represented explicitly.

We a design method and an operational semantics to efficiently handle diagnosis strategies. Within our framework we identify the class of one–step rule–like strategies, which is universal in the sense that it is expressive enough to describe every possible diagnostic process. Besides proving this theoretical result we demonstrate that all strategies needed in practice for our application domain can be denoted elegantly by one–step–strategies. Even preference concepts which have been modeled separately up to now [DS92, DPN94, FNS94] are modeled as one-step strategies. We show how to develop a strategic knowledge base for a specific application. Then we define an operational semantics which efficiently performs the diagnostic process using the strategies provided. In the course of the chapter we will discuss both deterministic and non–deterministic strategies for hierarchical circuit diagnosis. A major advantage of our method is that the strategies can be designed independently of each other, so that the strategic knowledge base can be easily extended without the need to rewrite existing strategies.

7.2 Working Hypotheses

We consider a system described by a set of formulas *SD* in a language $\mathcal{L}$. $\mathcal{L}$ is a first order language with equality. For simplicity we postulate that $\mathcal{L}$ contains no function symbols with variable interpretation. Functions with standard interpretation like mathematical operators etc. are allowed. By *ATOMS* we denote the set of atoms of $\mathcal{L}$. Our theory does not depend on a particular diagnosis definition. We encapsulate the underlying diagnosis concept by a function *diags*, which maps a theory T to a set of diagnoses $\mathcal{D}$:

$$diag(T) := \begin{cases} \emptyset, & \text{if } T \text{ is contradictory} \\ \begin{array}{l}\text{a set of diagnoses}\\ \mathcal{D} = \{D_1, \ldots D_n\},\end{array} & otherwise \end{cases}$$

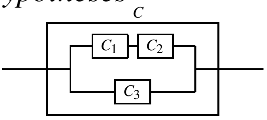
This general definition allows for a wide range of diagnosis concepts like minimal diagnoses [Rei87], most probable diagnoses [dKW87], preferred diagnoses [DPN94] and others.

In [Str92] Struss introduced the concept of working hypotheses into model–based diagnosis in order to make diagnostic assumptions explicit. The diagnostic assumptions are necessary in the diagnostic process for evaluating hierarchies, using simplified behavioral models, focusing on a particular kind of diagnoses, etc.

Definition 7.1 *Working Hypothesis*
Let $WHYP \subseteq ATOMS(\mathcal{L})$ be a defined set of atoms. Then each ground atom $wh \in WHYP$ is called a Working Hypothesis.

Working Hypotheses can be used to represent multiple models of the system within one system description as is shown by the following example.

Example 7.2 *Use of Working Hypotheses*



A component C consists of the subcomponents $C_1, \ldots, C_3$. The working hypothesis $refine(C)$ can be used to switch between the abstract model of C and the detailed model of C (in which its subcomponents $C_1, \ldots C_3$ are visible). In the system description SD for some device containing C the behavior of C is modeled depending on $refine(C)$:

$$\begin{array}{rcl} \neg refine(C) & \rightarrow & \textit{Rules for the abstract model of } C \\ refine(C) & \rightarrow & \textit{Rules for the detailed model of } C \end{array}$$

Now if we want to compute the diagnoses for the system based on the detailed model of C we add $refine(C)$ to the system description, i.e. we compute $diag(SD \cup OBS \cup \{refine(C)\})$.

In general, to compute the diagnoses under a set of working hypotheses s we add s to the system description. Additionally we add $\{\neg wh | wh \in WHYP \backslash s\}$, i.e. we make sure that s is exactly the set of working hypotheses which are true in the system model.

Definition 7.3 *Diagnosis under a Set s of Working Hypotheses*
Consider a system described by (SD, OBS) and a set of Working Hypotheses s. Let $\bar{s} = WHYP \backslash s$. Then the set of diagnoses under working hypotheses s, called $diag_s$ is defined as follows:

$$diag_s(SD \cup OBS) := \{D \cup s \mid s \in diag(SD \cup OBS \cup s \cup \{\neg wh | wh \in \bar{s}\})$$

We include the set s itself in the diagnosis, so that it is obvious from the diagnoses to which system model they belong. So $diag_s$ provides a valid diagnosis concept as $SD \cup OBS \cup diag_s(SD \cup OBS)$ is consistent.

Working hypotheses are an important concept for making the current diagnosis assumptions explicit. But the selection of suitable working hypotheses for a given situation is implicit in current diagnosis systems. In the next section we introduce a language that makes the knowledge for selecting the right hypotheses explicit and thus provides a flexible and declarative way of specifying strategic knowledge for the diagnostic process.

7.3 A Formal Language for Strategies

7.3.1 Preliminary Considerations

Diagnosis strategies control the diagnostic process by specifying which diagnosis assumptions should be used in a given situation. The state of the diagnostic process manifests itself in the current set of possible diagnoses. Therefore the specification of a diagnosis strategy consists of two parts:

- A property of the current set of diagnoses, characterizing a certain situation that can occur during the diagnostic process.

- An assumption or action (modeled by a working hypothesis) that is suitable for handling that situation

Example 7.4 *Diagnosis Strategy*
Consider an abstract component C as described in example 7.2. By default, we only use the abstract model of this component for diagnosis, i.e. $\neg refine(C)$ *is used as working hypothesis. The detailed model is only used when C is identified as faulty. This can be captured by the following rule:*

> *If an abstract component C occurs in all diagnoses, activate a more detailed model for C making its subcomponents visible*

In order to check if a given strategy should be applied we have to evaluate a condition on the current set of diagnoses. Such a condition can obviously not be modeled as part of the system description. So Boettcher and Dressler implement the check of these conditions as part of the diagnosis system.

However the only flexible way of evaluating these conditions is by introducing a meta–level in the diagnosis system as shown in figure 7.1. The strategies are defined on the meta level and they are evaluated using the knowledge obtained so far during the diagnostic process which is represented by the current possible diagnoses. The diagnoses $diag_s(SD \cup OBS)$ and the corresponding logical models are the information on which the decisions on the meta–level are based.

7.3.2 The Meta Language

The language $\mathcal{L}_{Strat}$ for defining diagnosis strategies defines modal logic operators specifying properties of the current diagnoses as well as for proposing working hypotheses. Before we give a formal definition of the language we motivate the need for these modal operators informally.

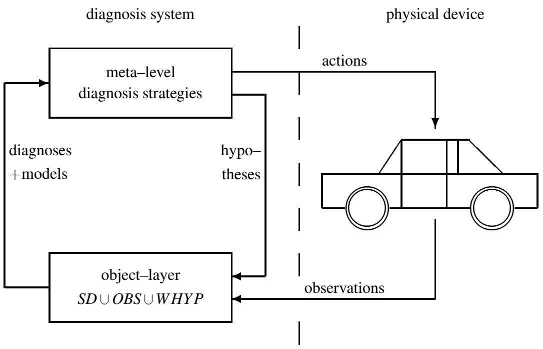


Figure 7.1: Diagnosis System with a Meta Level

Modal Operators for Characterizing the Current State of the Diagnostic Process:

As already stated the preconditions for the application of diagnosis strategies are statements about the current set of possible diagnoses. The atomic statements in these conditions are:

- a property $p(x)$ is true in all possible diagnoses, or
- a property $p(x)$ is true in at least one possible diagnosis.

These statements can be formalized using the usual S5 modal operators ($\Box$ for knowledge, $\Diamond$ for belief):

$\Box p$: p is true in all diagnoses of $SD \cup OBS \cup A$.

$\Diamond p$: p is true in at least one diagnosis of $SD \cup OBS \cup A$.

Modal Operators for Proposing Working Hypotheses

Strategy formulas specify which working hypotheses should be assumed in a given situation. This is achieved by the following (informally described) modal operators:

$\blacksquare\Box wh$: wh is a necessary working hypotheses in the current situation, i.e. the diagnostic process cannot be continued without assuming wh.

$\blacklozenge\Box wh$: wh is a possible (allowed) working hypotheses in the current situation.

We define a formal semantics for this language. In section 7.5 we define an algorithm, which constructs models of our formulas (denoting strategies) with certain desirable properties. Thus, we will use the semantics for model checking and model generation.

7.3.3 Syntax of the language

In general, the syntax is defined inductively as follows:

Definition 7.5 *$\mathcal{L}_{Strat}$–Formula*
Let $\mathcal{L}$ be a first-order language with equality.

1. *$L \in \mathcal{L}$ is a $\mathcal{L}_{Strat}$–Formula.*

2. *Let F, G be $\mathcal{L}_{Strat}$–Formulas and v, v_1, v_2 variables, then $\Box F$ and $\Diamond F$, $\blacksquare F$ and $\blacklozenge F$, $\neg F$ and $F \wedge G$, $v_1 = v_2$, $\forall v : F$ are $\mathcal{L}_{Strat}$–Formulas.*

3. *Nothing else is an $\mathcal{L}_{Strat}$–Formula.*

Note that $\mathcal{L}_{Strat}$ has the same predicate symbols and constants as the system description language $\mathcal{L}$. The variables denote objects of $\mathcal{L}$. This will be ensured in the formal semantics presented in the next section. The model-theoretic semantics will be given for the full language. In practice, we use a subset of the language $\mathcal{L}_{Strat}$, which can be efficiently handled by algorithms. It turned out that rule-like strategies are best suited to express intuition: Based on properties of the current state of the diagnostic process the strategies propose new working hypotheses. Such strategies are restricted $\mathcal{L}_{Strat}$–formulas which can be written as a rules $C \rightarrow H$ where C characterizes the current diagnostic state and H the immediate successor states. The head H has one level of modalities $\blacklozenge$ and $\blacksquare$, the body C has none. Such *One-Step Strategies* are rules where the body has depth 0 and the head depth 1 wrt. to the bold modalities. Since we deal only with one-step strategies in this chapter we simply call them strategies and explicitly refer to $\mathcal{L}_{Strat}$–formulas, when we need the full language.

Definition 7.6 *Depth*
If $L \in \mathcal{L}$, then $\Box L$ and $\Diamond L$ are formulas of depth 0. Let F, G be both formulas of depth n, then $\Box F$, $\Diamond F$, $\neg F$, $F \wedge G$, $F \vee G$ are formulas of depth n, $\blacksquare F$ and $\blacklozenge F$ are formulas of depth $n+1$.

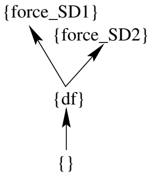
Definition 7.7 *One–Step Strategy*
A One–step Strategy *(in this chapter simply called* Strategy*) is a formula of the form $C \rightarrow H$, where C is a formula of depth 0 and H is a formula of depth 1.*

7.3.4 Representation of a Diagnostic Process

The aim of strategies is to control the diagnostic process by proposing suitable working hypotheses. The diagnostic process itself can be non–deterministic, because more than one set of hypotheses can be proposed for a given situation. The *State* of the diagnostic process is characterized by the current set of working hypotheses. For representing diagnostic processes we use the notion of a *State Transition System*:

Definition 7.8 *State Transition System*
Let S be a finite set of states, $t \in S$ an initial state and $\rightarrow \subseteq S \times S$ a transition relation. Then $(S, \rightarrow, t)$ is called a State Transition System. *We assume $S \subseteq 2^{WHYP}$.*

Consider the following diagnostic process. We start the diagnosis with a simple description of a device (SD_0) and the single–fault–assumption.[1] It turns out that no single–fault diagnosis exists, thus we allow the assumption of double faults. Again no diagnoses are found. Now we consider the simple description of the device too abstract and we propose the activation of one of the more detailed descriptions SD_1 or SD_2. In both of these system descriptions we would find single–fault diagnoses.



The transition system for representing this process has the states $\emptyset$ (initial state, no working hypotheses), df (then allow double faults), *force_SD1* and *force_SD2* (select one of the more detailed system descriptions).

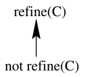
7.3.5 Designing Strategies

The diagnostic process is represented by a transition system. We use strategies to influence the shape of this transition system. Let us first show how to define strategies in order to obtain a certain transition system.

Deterministic Strategies.

Often we want to assume one specific hypothesis in a given situation. For example, if we have found that a component C is definitely faulty, we activate the refined description for it to see which of its subcomponents caused the fault. We need a strategy which proposes a state transition leading to a state in which *refine*(C) holds:

$$\Box ab(C) \rightarrow \blacklozenge\Box refine(C) \wedge \blacksquare\Box refine(C)$$



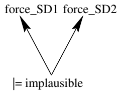
The formula describes exactly the transition system wrt. the hypothesis *refine(C)*. Other strategies specify transition systems wrt. other hypotheses. These partial transition systems are then combined by the algorithm which computes the diagnostic process. The $\blacklozenge$–operator is necessary in this formula. If we had only used the $\blacksquare$–operator in the conclusion of this formula, the formula would have been satisfied also if there were no successor of the current state. The quantification over ”all successor states” would then be trivially satisfied.

[1] In section 7.4 we show how theses assumptions can be modeled by strategies.

Non–Deterministic Strategies.

Sometimes there are several possibilities for continuing the diagnostic process in a given situation. Again we can first develop a transition system and then describe it by a strategy.

$$\begin{aligned}&\Box \mathit{implausible} \rightarrow \\ &\quad \blacklozenge\Box \mathit{force\_M1} \wedge \blacklozenge\Box \mathit{force\_M2} \\ &\quad \wedge \blacksquare(\Box \mathit{force\_M1} \not\leftrightarrow \Box \mathit{force\_M2})\end{aligned}$$



In this strategy *implausible* is a predicate in the system description, which indicates that there is no plausible diagnosis under the current system description.

Example 7.9 *Implausible*
A diagnosis may be implausible if it contains triple faults including incomplete faults. There are triple faults if three distinct components X,Y,Z are abnormal and at least one of them is in an unknown fault mode. We denote the fault mode M of a component C by $fm(C,M)$.

$$\begin{aligned}\forall X,Y,Z: \quad & ab(X) \wedge ab(Y) \wedge ab(Z) \wedge \\ & X \neq Y \wedge X \neq Z \wedge Y \neq Z \wedge \\ & (fm(X,unknown) \vee fm(Y,unknown) \vee fm(Z,unknown)) \\ & \qquad \rightarrow implausible\end{aligned}$$

The strategy for the choice of system description proposes two possibilities for continuing the diagnostic process: Either use SD_1 or use SD_2 but do not use both descriptions at the same time. We will give a more specific account on the selection of an appropriate system description in section 7.4.

The above design method allows to define strategies independently without having to care about interference between different strategies. We will use it to define a complete set of strategies for circuit diagnosis (section 7.4) and describe how the independence can be preserved during the diagnostic process (section 7.5).

7.3.6 Consistency of Transition Systems

In order to check if the decisions made during the diagnostic process are consistent with the given diagnosis problem, we define how strategies can be interpreted as statements describing the diagnostic process. First, we define logical structures which provide the interpretation for the strategies.

Definition 7.10 *$\mathcal{L}_{Strat}$–Model*
A model for $\mathcal{L}_{Strat}$ is a structure $M = \langle W, D, \rightarrow_1, \rightarrow_2, I\rangle$, where W is a set of individuals (called worlds), D is a domain of individuals, $\rightarrow_1$ and $\rightarrow_2$ are accessibility relations on the worlds, i.e. subsets of $W \times W$, and I is an interpretation function.

I provides the interpretation for predicates and ground terms (in our case all ground terms are constants). The values of the variables are given by an assignment:

Definition 7.11 *Assignment*
Given a domain D an Assignment *is a mapping from the set of variables into D. By* $\alpha_{(x|d)}$ *we denote the assignment that maps variable x to an element $d \in D$ and is defined in the same way as* α *on the other variables.*

The semantics of an $\mathcal{L}_{Strat}$–Model is defined as follows:

Definition 7.12 *Semantics of* $\mathcal{L}_{Strat}$
Let $M = (W, D, \rightarrow_1, \rightarrow_2, I)$ be an $\mathcal{L}_{Strat}$–model, F, G $\mathcal{L}_{Strat}$–Formulas, α *an assignment and $w \in W$. Let*

$$Val(t,\alpha,w) := \begin{cases} \alpha(t), & \textit{iff } t \textit{ is a variable} \\ I(w,t), & \textit{iff } t \textit{ is a constant} \end{cases}$$

Then

$$\begin{array}{lll}
M \models_{w,\alpha} P(t_1, \ldots t_n) & \textit{iff} & (Val(t_1,\alpha,w), \ldots Val(t_n,\alpha,w)) \\
 & & \in I(w,P) \\
M \models_{w,\alpha} t_1 = t_2 & \textit{iff} & Val(t_1,\alpha,w) = Val(t_2,\alpha,w) \\
M \models_{w,\alpha} F \wedge G & \textit{iff} & M \models_{w,\alpha} F \textit{ and } M \models_{w,\alpha} G \\
M \models_{w,\alpha} \neg F & \textit{iff} & M \not\models_{w,\alpha} F \\
M \models_{w,\alpha} \blacksquare F & \textit{iff} & \textit{f.a. } w' \in W \textit{ s.th. } w \rightarrow_1 w' \textit{: } M \models_{w',\alpha} F \\
M \models_{w,\alpha} \blacklozenge F & \textit{iff} & \textit{ex. } w' \in W \textit{ s.th. } w \rightarrow_1 w' \textit{ and } M \models_{w',\alpha} F \\
M \models_{w,\alpha} \Box F & \textit{iff} & \textit{f.a. } w' \in W \textit{ s.th. } w \rightarrow_2 w' \textit{: } M \models_{w',\alpha} F \\
M \models_{w,\alpha} \Diamond F & \textit{iff} & \textit{ex. } w' \in W \textit{ s.t. } w \rightarrow_2 w' \textit{ and } M \models_{w',\alpha} F \\
M \models_{w,\alpha} \forall x.F & \textit{iff} & \textit{f.a. } d \in D : M \models_{w,\alpha_{(x|d)}} F
\end{array}$$

We will use the following abbreviations:

$$\begin{array}{lll}
M \models_w F & \textit{iff} & M \models_{w,\alpha} F \textit{ for every assignment } \alpha. \\
M \models F & \textit{iff} & M \models_w F \textit{ for every } w \in W
\end{array}$$

The connection between the state transition relation and the $\mathcal{L}_{Strat}$–model is established in the following way: The possible diagnoses are interpreted as possible worlds, where all the diagnoses under one set of working hypotheses are connected wrt. the $\Box$–operator (relation $\rightarrow_2$). The accessibility relation for the $\blacksquare$–Operator (relation $\rightarrow_1$) is given by the state transition relation $\rightarrow$, i.e. diagnoses under different working hypotheses are connected by $\rightarrow_1$ iff the underlying sets of working hypotheses are connected by $\rightarrow$.

Definition 7.13 *Induced $\mathcal{L}_{Strat}$–Model $M_{(S,\rightarrow,t)}$*
Let $(S, \rightarrow, t)$ be a state transition system. For $s \in S$ let $\mathcal{D}_s = diag_s(SD \cup OBS)$ be the diagnoses under s. We distinguish two cases:

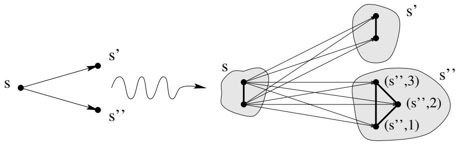


Figure 7.2: Example of the two accessibility relations

1. *If $\mathcal{D}_s \neq \emptyset$, then let m_s be the number of models obtained from the system description, the observations and the diagnoses in $\mathcal{D}_s$ and let $\{M_{(s,1)},\dots,M_{(s,m_s)}\}$ be the corresponding set of Herbrand models.*

2. *If $\mathcal{D}_s = \emptyset$, then let $m_s = 1$ and $M_{(s,1)}$ be the Herbrand model defined by $M_{(s,1)} = s \cup \{ab(SD)\}$*

The Induced $\mathcal{L}_{Strat}$–Model $M_{(S,\rightarrow,t)}$ *is defined as*

$$\begin{array}{rcl} M_{(S,\rightarrow,t)} & = & \langle W', D', \rightarrow'_1, \rightarrow'_2, I' \rangle, \textit{ where} \\ & & \\ W' & = & \{(s,i) \mid s \in S, i \in \{1,\dots,m_s\}\} \\ D' & & \textit{is the set of constants in } \mathcal{L} \\ \rightarrow'_1 & = & \{((s,i),(s',j)) | s \rightarrow s', 1 \leq i \leq m_s, 1 \leq j \leq m_{s'}\} \\ \rightarrow'_2 & = & \{((s,i),(s,j)) \mid s \in S, i,j \in \{1,\dots,m_s\}\} \\ I'((s,i),P) & = & \{\vec{x} \in \{D'^n | P(\vec{x}) \in M_{(s,i)}\} \textit{ for a predicate symbole } P \textit{ of} \\ & & \textit{arity } n \\ I'((s,i),a) & = & a\textit{, for a constant } a \textit{ in } \mathcal{L}. \end{array}$$

Definition 7.14
Let F be an $\mathcal{L}_{Strat}$–Formula and $(S,\rightarrow,t)$ a transition system with induced model $M_{(S,\rightarrow,t)}$ and $s \in S$. We write $(S,\rightarrow,t) \models_s F$ iff $M_{(S,\rightarrow,t)} \models_{(s,i)} F$.

The induced model needs some explanation. The worlds W' are a set of references to models which are either obtained from diagnoses of the system description and observations under a set of working hypothesis or given by $M_{(s,1)} = s \cup \{ab(SD)\}$ in case there are no diagnoses. There are two transition relations. The first one, $\rightarrow'_1$, is inherited from the given state transition system $(S,\rightarrow,t)$. For each transition $s \rightarrow s'$ from a set of working hypotheses s to another one s' there are transitions $(s,i) \rightarrow'_1 (s',j)$ where i and j are needed to identify the corresponding models $M_{(s,i)}$ and $M_{(s',j)}$. The second transition relation connects all models obtained under the same set of working hypotheses.

Example 7.15 *Induced Model*
The transition system on the left of Figure 1 induces the model on the right. For each set s of working hypotheses a set of models is obtained. The relation $\rightarrow'_2$ which is

shown in bold arcs connects all models obtained under a set s of working hypotheses. The relation $\rightarrow$ on the left induces the transitions $\rightarrow_1'$ on the right.

To understand the declarative semantics, consider a given transition system $(S, \rightarrow, t)$. We check whether this transition system is a valid solution to the diagnostic problem given by $SD \cup OBS$ and a set of strategies $\mathcal{F}$. We call a transition system *consistent*, iff all its states satisfy all formulas from $\mathcal{F}$. Intuitively, the way the diagnostic process proceeds is consistent with what the strategies propose.

Definition 7.16 *Consistent*
A transition system $(S, \rightarrow, t)$ is consistent *with a set $\mathcal{F}$ of strategies, iff for all formulas $F \in \mathcal{F}$ and all $s \in S$ we have $(S, \rightarrow, t) \models_s F$.*

An important step to compute a consistent transition system is given by the next proposition. For the modalities $\Box$ and $\Diamond$ consider first a formula $L \in \mathcal{L}$ without modal operators. A state s satisfies $\Diamond L$ if there is a diagnosis corresponding to state s under which L holds, and a state satisfies $\Box L$, if L holds under all diagnoses in state s. Additionally, we consider the case when there are no diagnoses, because the inconsistency of certain assumptions with the current situation should not lead to termination of the diagnostic process but rather to a change of assumptions. The absence of diagnoses under a given set of literals is indicated by the literal $ab(SD)$, intuitively indicating that the system description is not suited for the current set of assumptions.

For the modalities $\blacksquare$ and $\blacklozenge$ let F be a strategy formula. The semantics for the operators $\blacksquare F$ and $\blacklozenge F$ is given by the transitions. A state s satisfies $\blacksquare F$, if the formula F holds in all successor states. Similarly, a state satisfies $\blacklozenge F$, if the formula F holds in at least one successor state.

Proposition 7.17 *Let $(S, \rightarrow, t)$ be a transition system, $s \in S$, and $\mathcal{D}_s$ be the diagnoses under s. Let $L \in \mathcal{L}$ be a first order formula and F a strategy, then*

$$
\begin{array}{lll}
(S,\rightarrow,t) \models_s \Diamond L, & \textit{iff} & \mathcal{D}_s \neq \emptyset \textit{ and there is a diagnosis } D \in \mathcal{D}_s \textit{ such} \\
 & & \textit{that the model for } SD \cup OBS \cup s \cup \neg\bar{s} \cup D \textit{ entails } L \\
 & \textit{or} & \mathcal{D}_s = \emptyset \textit{ and the model for } s \cup \neg\bar{s} \cup \{ab(SD)\} \\
 & & \textit{entails } L \\
(S,\rightarrow,t) \models_s \Box L, & \textit{iff} & \mathcal{D}_s \neq \emptyset \textit{ and for all } D \in \mathcal{D}_s\textit{: } SD \cup OBS \cup s \cup \\
 & & \neg\bar{s} \cup D \textit{ entails } L \\
 & \textit{or} & \mathcal{D}_s = \emptyset \textit{ and } s \cup \neg\bar{s} \cup \{ab(SD)\} \textit{ entails } L. \\
(S,\rightarrow,t) \models_s \blacklozenge F, & \textit{iff} & \textit{ex. } s' \in S \textit{ such that } s \rightarrow s' \textit{ and } (S,\rightarrow,t) \models_{s'} \\
 & & F. \\
(S,\rightarrow,t) \models_s \blacksquare F, & \textit{iff} & \textit{f.a. } s' \in S\textit{: from } s \rightarrow s' \textit{ follows } (S,\rightarrow,t) \models_{s'} \\
 & & F.
\end{array}
$$

Proof:

1. We show that $(S,\rightarrow,t) \models_s \Diamond L$ iff $\mathcal{D} \neq \emptyset$ and there is a diagnosis $D \in \mathcal{D}_s$ such that the model for $SD \cup OBS \cup s \cup \neg\bar{s} \cup D$ entails L or $\mathcal{D} = \emptyset$ and the model for $s \cup \neg\bar{s} \cup \{ab(SD)\}$ entails L.

   Before we start the proof we need a lemma

   Lemma 7.18 *Let $L \in \mathcal{L}$. Then $M_{(S,\rightarrow,t)} \models_{(s,j)} L$ iff $M_{(s,j)} \models_{\mathcal{L}} L$.*

   By definition 7.14 $(S,\rightarrow,t) \models_s \Diamond L$ iff $M_{(S,\rightarrow,t)} \models_{(s,i)} \Diamond L$. By definition 7.12 this is equivalent to $\exists w'$ s.th. $(s,i) \rightarrow_2' w'$ and $M_{(S,\rightarrow,t)} \models_{w'} L$. By definition 7.13 there is $1 \leq j \leq m_s$ s.th. $(s,i) \rightarrow_2' (s,j)$ and $M_{(S,\rightarrow,t)} \models_{(s,j)} L$ and by the lemma above we have $M_{(s,j)} \models_{\mathcal{L}} L$ which means by the definition of $M_{(s,j)}$ in definition 7.13 that $\mathcal{D} \neq \emptyset$ and there is a diagnosis $D \in \mathcal{D}_s$ such that the model for $SD \cup OBS \cup s \cup \neg\bar{s} \cup D$ entails L or $\mathcal{D} = \emptyset$ and the model for $s \cup \neg\bar{s} \cup \{ab(SD)\}$ entails L.

2. The proof for $\Box L$ is similar.

3. To prove $(S,\rightarrow,t) \models_s \blacklozenge F$ iff ex. $s' \in S$ such that $s \rightarrow s'$ and $(S,\rightarrow,t) \models_{s'} F$ we proceed as above and apply definition 7.14, 7.12, and 7.13.

4. The proof for $\blacksquare F$ is similar. Q.E.D.

7.3.7 Results of the Diagnostic Process

The aim of the diagnostic process could be to identify one unique diagnosis. In general this would be a too restrictive criterion for terminating the diagnostic process because we might not have enough knowledge to discriminate among all the diagnoses. So we define that the diagnostic process terminates in a state where we assumed exactly all possible and necessary hypotheses.

This corresponds to a loop in the transition system as depicted on the right. If such a state yields diagnoses we cannot reach a more preferred state by applying another strategy.

Definition 7.19 *Stable State*
Let s be a state in the consistent state transition system $(S,\rightarrow,t)$ and $\mathcal{F}$ a set of strategy formulas. The state s is stable *wrt. $(S,\rightarrow,t)$, iff*

1. $diag_s(SD,OBS) \neq \emptyset$

2. $s = \{wh \mid (S,\rightarrow,t) \models_s \blacklozenge\Box wh \wedge \blacksquare\Box wh\}$

It is called weakly stable*, if $s \subseteq \{wh \mid (S,\rightarrow,t) \models_s \blacklozenge\Box wh \wedge \blacksquare\Box wh\}$*

The first condition states that $SD \cup OBS \cup s$ is consistent and the second condition is a fixpoint condition: s is already the set of all possible and necessary working hypotheses. The result of the diagnostic process is given by the diagnoses corresponding to the stable states and weakly stable states, respectively.

7.4 A Strategy Knowledge Base for Circuit Diagnosis

In this section we apply the strategy language to the diagnosis of digital circuits. We model strategies such as the choice among multiple system descriptions, structural refinement, measurements, and preferences. In section 7.6 we use them to diagnose the voter circuit from [Isc85].

Multiple Views.

Multiple views allow to describe the diagnosed systems emphasizing different aspects. For circuit diagnosis it is often important to consider a physical view beside a functional one, because the physical view additionally takes the layout into account [Dav84]. We want to employ the functional view by default and the physical view only if we do not obtain good diagnoses.

Strategy (1) tells us how to choose between the views using the hypotheses *force_physical* and *force_functional*. The predicate *implausible*, which in our example holds if no single or double fault diagnosis exists, indicates that the other view should be activated. To avoid more than one activation it is also checked that *force_functional* does not yet hold. Once the body of the strategy is satisfied we have to make sure that the diagnostic process continues in two directions with the functional and the physical view, respectively, as active model. Thus, we adopt either *force_functional* or *force_physical*. Once the selection of the view has taken place both hypotheses are kept by monotonic addition of working hypotheses $(2,3)$.

$$\begin{aligned}
&\Box \mathit{functional} \wedge \Box \mathit{implausible} \wedge \Box\neg \mathit{force\_functional} \rightarrow \\
&\quad \blacklozenge\Box \mathit{force\_functional} \wedge \blacklozenge\Box \mathit{force\_physical} \wedge \\
&\quad \blacksquare\Box(\mathit{force\_functional} \leftrightarrow \neg \mathit{force\_physical}) && (1)\\
&\Box \mathit{force\_physical} \rightarrow \blacklozenge\Box \mathit{force\_physical} \wedge \blacksquare\Box \mathit{force\_physical} && (2)\\
&\Box \mathit{force\_functional} \rightarrow \blacklozenge\Box \mathit{force\_functional} \wedge \blacksquare\Box \mathit{force\_functional} && (3)
\end{aligned}$$

In the system description we model the connection between the working hypotheses *force_physical* and *force_functional* and the literals *physical* and *functional*, which select the appropriate system description. The functional view is used by default when no hypothesis is active:

$$\begin{gathered}
\neg \mathit{force\_functional} \wedge \neg \mathit{force\_physical} \rightarrow \mathit{functional} \\
\mathit{force\_functional} \rightarrow \mathit{functional},\ \mathit{force\_physical} \rightarrow \mathit{physical}
\end{gathered}$$

Structural Refinement.

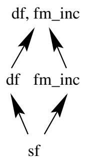
Many authors address the use of hierarchies to reduce the complexity of diagnosis problems [Dav84, Ham91, Gen84, Moz91, BD94]. In particular, Böttcher and Dressler introduce the strategy of structural refinement which states that an abstract description of a component is refined only if it is uniquely identified as defective [BD94]. Only if all diagnoses contain a component C, it is possible and necessary to activate a detailed description of C:

$$\begin{aligned} &\forall C.(\Box ab(C) \wedge \mathit{refineable}(C)) \vee \Box \mathit{refine}(C) \rightarrow \\ &\quad \blacklozenge \Box \mathit{refine}(C) \wedge \blacksquare \Box \mathit{refine}(C) \end{aligned} \tag{4}$$

In the system description the rules of the abstract description are active when $\mathit{refine}(C)$ is false and the rules of the detailed description are activated if $\mathit{refine}(C)$ is true. This variant of using hierarchies is very efficient since the refinement of the system description is postponed until the erroneous components are identified.

Preference Relations among Diagnoses.

Preferences state that diagnoses with certain properties are better than others with other properties [DS92, DPN94, FNS94]. Frequently used preferences are for example the single fault assumption or "physical negation" [SD89], i.e. the assumption that the known fault models of the components are complete. To use preferences efficiently, the preferred property is activated by default and is relaxed only if there are no diagnoses which have the intended property. We can use $ab(SD\ )$ to detect if there are any diagnoses. $ab(SD\ )$ holds iff $diag_s(SD \cup OBS) = \emptyset$ (see sec. 7.3.6).



The preference relation on the left states that by default we are only interested in single faults (*sf*). If there are no diagnoses under the single–fault assumption we allow either diagnoses with double faults (*df*) or incompleteness of the fault models (*fm_inc*). If there are still no diagnoses under one of these relaxed hypotheses, we allow double fault diagnoses and incompleteness of fault models at the same time.

Example 7.20 *sf and df*
A working hypothesis nf (for n-faults) restricts the cardinality of diagnoses. In the system description it is used as a precondition in a formula which restricts the number of faults (ab–literals). For single and double faults we have, for example

$$\begin{array}{rl} \forall C_1, C_2: & sf \wedge ab(C_1) \wedge ab(C_2) \rightarrow C_1 = C_2 \\ \forall C_1, C_2, C_3: & df \wedge ab(C_1) \wedge ab(C_2) \wedge ab(C_3) \rightarrow \\ & \qquad C_1 = C_2 \vee C_1 = C_3 \vee C_2 = C3 \end{array}$$

Proposition 7.21 *n–faults*
Single- and double–fault assumption can be generalized to n–faults.

$$\forall C_1, \ldots, C_{n+1} : nf \wedge \bigwedge_{i=1}^{n+1} ab(C_i) \rightarrow \bigvee_{i,j=1, i \neq j}^{n+1} C_i = C_j$$

If SD contains the above formula and $nf \in s$ for some n, then for any diagnosis $D \in diag_s(SD \cup OBS)$ there are no more than n distinct components C_i s.th. $ab(C_i) \in D$.

Proof: We show that any diagnosis under the n-fault assumption nf does not contain more than n components. Let $\mathcal{D}_s = diag_s(SD \cup OBS)$. Assume $nf \in s$ and $D \in \mathcal{D}_s$ s.th. $|D| > n$. Then there are at least $n+1$ distinct components C_i s.th. $ab(C_i) \in D$. Thus the lefthand side of the formula

$$\forall C_1, \ldots, C_{n+1} : nf \wedge \bigwedge_{i=1}^{n+1} ab(C_i) \rightarrow \bigvee_{i,j=1, i \neq j}^{n+1} C_i = C_j$$

is satisfied. To be consistent the right-hand side has to be satisfied as well, i.e. at least two components are equal. This is contradictory to choosing $n+1$ distinct components. Q.E.D.

The system description captures the default assumption of single faults by the rule $\neg df \wedge \neg tf \rightarrow sf$. The strategy of relaxing the single-fault property (5) checks if no diagnoses exist (using $ab(SD)$) and if neither df nor fm_inc hold. In this case it is possible to adopt either fm_inc or df, but not both at the same time. Finally, double faults together with the assumption of incomplete fault models are allowed only if there are no double fault diagnoses and no single–fault diagnoses with incomplete fault modes (6).

$$\begin{aligned}
&\Box ab(SD) \wedge \Box \neg df \wedge \Box \neg fm\_inc \rightarrow \\
&\quad \blacklozenge \Box fm\_inc \wedge \blacklozenge \Box df \wedge \blacksquare(\Box df \not\leftrightarrow fm\_inc) \qquad (5) \\
&\Box ab(SD) \wedge (\Box df \vee \Box fm\_inc) \rightarrow \\
&\quad \blacklozenge \Box(df \wedge fm\_inc) \wedge \blacksquare \Box(df \wedge fm\_inc) \qquad (6)
\end{aligned}$$

These strategies show the desired behavior discussed and implemented in [FNS94]. First, the diagnosis system tries to find diagnoses under the most preferred set of properties (in our case diagnoses with only single faults). Only if this is not possible (i.e. $ab(SD)$) is true), these properties are exchanged with the next most preferred set of properties (in our case either double faults or the assumption of “fault mode incomplete”) and so on. With respect to other strategies the preference properties are not monotonic: Whenever a diagnosis strategy not related to these preferences is executed (e.g. refinement, multiple views, etc.), diagnosis in this changed state again starts by trying to find diagnoses corresponding to the most preferred set of properties.

Measurements.

De Kleer, Raiman and Shirley view diagnosis as an incremental task involving the three phases of generating explanations, choosing actions differentiating among them and performing these actions [dKRS91]. Our framework allows to incorporate these phases into the diagnostic process. Strategy (7) proposes a point X of the circuit for measurement if there are two diagnoses predicting different values for X. As measurements are expensive, we want to apply the strategy only if all refinements are already done which is checked in the first line.

$$\begin{aligned}
&\forall C.(\Box \mathit{refineable}(C) \wedge \Diamond \mathit{ab}(C) \rightarrow \Box \mathit{refine}(C)) \wedge \\
&\forall X.\Box \mathit{point}(X) \wedge \Diamond \mathit{val}(X,0) \wedge \Diamond \mathit{val}(X,1) \rightarrow \\
&\quad \blacklozenge \Box \mathit{propose}(X) \wedge \blacksquare \Box \mathit{propose}(X) \qquad (7)
\end{aligned}$$

The second phase of choosing the right action is carried out by procedural attachment in the system description. The (procedural) predicate *best_meas(X)* is true for a measurement point X, which is optimal according to some specification (for example minimum entropy). It only needs to be evaluated for the measurement points X proposed by strategy (8).

$$\begin{aligned}
&\forall X.\Box \mathit{propose}(X) \wedge \Box \mathit{best\_meas}(X) \rightarrow \\
&\quad \blacklozenge \Box \mathit{measure}(X) \wedge \blacksquare \Box \mathit{measure}(X) \qquad (8)
\end{aligned}$$

In the system description the hypothesis $\mathit{measure}(X)$ causes the specific measurement of X to be executed, which is also done by procedural attachment. The modification of the system description due to the insertion of the measured value has to be reflected by a change of the diagnostic state. This is achieved by using the hypothesis *measure(X)* in a monotonic way (9). Another weak monotonicity axiom is used for *propose* which is active until the next consistent state is reached in which the measurement is carried out, i.e. as long as $\mathit{ab}(\mathit{SD})$ holds (10).

$$\begin{aligned}
&\forall X.\Box \mathit{measure}(X) \rightarrow \blacklozenge \Box \mathit{measure}(X) \wedge \blacksquare \Box \mathit{measure}(X) \qquad (9) \\
&\forall X.\Box \mathit{ab}(\mathit{SD}) \wedge \Box \mathit{propose}(X) \rightarrow \\
&\quad \blacklozenge \Box \mathit{propose}(X) \wedge \blacksquare \Box \mathit{propose}(X) \qquad (10)
\end{aligned}$$

7.5 Operational Semantics

The strategies presented in this chapter have been designed by describing transition systems. These strategies have the important property that they are satisfied by exactly one transition system. Thus, the meaning of these strategies is completely determined by the semantics of the strategy language. Now the question arises, whether every transition system can be defined by a strategy. The answer is positive. We will define the *Characteristic Formula* of a transition system in this section. All the strategies in this chapter are equivalent to characteristic formulas. We further present a method which combines the transition systems defined by a set of strategies. Our method will have the property that it maximizes the chance that a consistent diagnostic process is found. Finally, the method is exploited by a simple iterative algorithm.

Given a transition system $(S, \rightarrow, t)$, we can systematically define a formula, which completely characterizes $(S, \rightarrow, t)$:

Definition 7.22 *Characteristic Formula*
Let $(S, \rightarrow, t)$ be a transition system, WHYP a set of working hypotheses and $s \in S$. Then G_s is called State Formula *of s*, *F_s* is called *Characteristic Formula of s and F_t is called* Characteristic Formula *of $(S, \rightarrow, t)$, where*

$$\begin{array}{rcl} G_s & = & \Box\bigwedge\{wh \mid wh \in s\} \cup \{\neg wh \mid wh \in (\bigcup S)\backslash s\} \\ F_s & = & G_s \wedge \bigwedge_{s \to s'} \blacklozenge F_{s'} \wedge \blacksquare \bigvee_{s \to s'} F_{s'} \end{array}$$

Lemma 7.23 *Let $(S, \to, \emptyset)$ be a transition system and $s, s' \in S$ then $(S, \to, \emptyset) \models_s G_{s'}$ iff $s = s'$.*

Proof: $\Leftarrow$ Application of the definition of $\models_s$ and G_s.
$\Rightarrow$ Assume $s \neq s'$. Then there is a working hypothesis wh such that without loss of generality $wh \in s'$ but $wh \notin s$. From the premise $(S, \to, \emptyset) \models_{s'} G_s$ we can conclude $(S, \to, \emptyset) \models_{s'} \Box\neg wh$ and $(S, \to, \emptyset) \not\models_{s'} \Box wh$ since $wh \notin s$. But by definition of $\models_{s'}$ we have also $(S, \to, \emptyset) \models_{s'} \Box wh$ since $wh \in s'$. A contradiction. Q.E.D.

Note that the last conclusion requires that a working hypothesis is only derivable iff it is contained in the state which is guaranteed by adding working hypotheses and the negated complements.

The above lemma states that the state formula G_s holds only at state s and is false at any other state. Thus, the first conjunct G_s of the formula F_s *fully* characterizes the current state s, the second conjunct manifests the existence of the successor states and the third conjunct states that there are no other successors.

Example 7.24 *For the strategy structural refinement in section 7.4, the head of the strategy is a characteristic formula for the transition system with the only transition from the empty set to the state $\{refine(C)\}$.*

In general, all the strategy heads in this chapter are equivalent to characteristic formulas. The following theorem shows that we can uniquely characterize a given transition system by a set of strategies.

Theorem 7.25 *Given a transition system $(S, \to, \emptyset)$, then there is a set $\mathcal{F}$ of one-step strategies, such that $(S', \to', \emptyset) \models \mathcal{F}$ iff $(S', \to', \emptyset) = (S, \to, \emptyset)$.*

Proof: The set $\mathcal{F}$ of one-step strategies is defined by transition formulas that express that if we are in state s, where G_s holds then the successors' state formulas hold possibly and their disjunction holds necessarily. The case that the state s has no successors is captured by the formula $\blacksquare$*false*:

Definition 7.26 *Let $(S, \to, \emptyset)$ be a transition system and G_s the state formula for a state $s \in S$. Then*

$$\begin{array}{ll} G_s \to \bigwedge_{s \to s'} \blacklozenge G_{s'} \wedge \blacksquare \bigvee_{s \to s'} G_{s'} & \textit{if there is } s'' \textit{ such that } s \to s'' \\ G_s \to \blacksquare \textit{false} & \textit{else} \end{array}$$

is called Transition Formula of *s*.

Remark: Transition formulas of states with successors are one-step strategies. The heads are characteristic formulas.

Now we can turn to the proof of the theorem. Let $(S,\rightarrow,\emptyset)$ be a transition system and let $\mathcal{F}$ be the set of transition formulas of the states S.

$\Leftarrow$ We have to show that $(S,\rightarrow,\emptyset) \models \mathcal{F}$. Let $s \in S$ and $F_s \in \mathcal{F}$. We show that for all states $s' \in S$ we have $(S,\rightarrow,\emptyset) \models_{s'} F_s$ by examining the cases $s \neq s'$ and $s = s'$. In case $s' \neq s$ we know by lemma 7.23 that $(S,\rightarrow,\emptyset) \not\models_{s'} G_s$ and thus $(S,\rightarrow,\emptyset) \models_{s'} F_s$. In case $s' = s$ we know by lemma 7.23 that $(S,\rightarrow,\emptyset) \models_{s'} G_s$ so that it is left to prove that $(S,\rightarrow,\emptyset) \models_{s'} \bigwedge\limits_{s\rightarrow s''} \blacklozenge G_{s''} \wedge \blacksquare \bigvee\limits_{s\rightarrow s''} G_{s''}$ in case $\exists s'' : s \rightarrow s''$ and $(S,\rightarrow,\emptyset) \models_{s'} \blacksquare \mathit{false}$ otherwise. The latter formulas hold by definition of $\blacksquare$ and $\blacklozenge$.

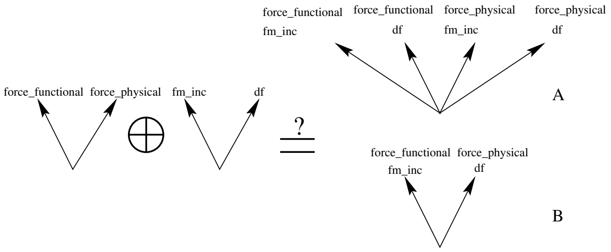
$\Rightarrow$ The proof is by contradiction. Assume that $(S',\rightarrow',\emptyset) \models \mathcal{F}$ and $(S',\rightarrow',\emptyset) \neq (S,\rightarrow,\emptyset)$. We have $\emptyset \in S$ and starting with $\emptyset$ all states of S are "traversed" by the transition formulas $\mathcal{F}$. Since also $\emptyset \in S'$ and $(S',\rightarrow',\emptyset) \models \mathcal{F}$ we can conclude that $S' \supset S$. Now assume $s \in S' \cap S$ and $s'' \in S' \backslash S$ such that $s \rightarrow s''$. Since $s'' \notin S$ we have $(S',\rightarrow',\emptyset) \not\models_{s''} \bigvee\limits_{s\rightarrow s'} G_{s'}$. As $s \rightarrow s''$ we conclude $(S',\rightarrow',\emptyset) \not\models_{s} \blacksquare \bigvee\limits_{s\rightarrow s'} G_{s'}$. From this and the fact that $(S',\rightarrow',\emptyset) \models_s G_s$ we conclude $(S',\rightarrow',\emptyset) \not\models_s F_s$ in contradiction to the assumption. Q.E.D.

We conclude that one-step strategies are the "smallest" language that captures all possible transition systems, because we need at least one level of $\blacksquare$ and $\blacklozenge$–Modalities to describe transitions and we have now shown that one level is also sufficient to describe a given transition system.

7.5.1 Combining Strategies

When we consider more than one strategy formula we have to solve the problem of combining the proposed transitions. Suppose we have two strategy formulas $C_1 \rightarrow H_1$ and $C_2 \rightarrow H_2$ and the current state of the process satisfies both C_1 and C_2. How do we combine the transitions proposed by H_1 and H_2?

Example 7.27 *Recall the strategies* (5) *of preferring single fault diagnoses over double faults and incomplete fault modes and the strategy* (1) *of activating the physical view. Assume the bodies of* (5) *and* (1) *are satisfied in the current state and we have to perform transitions to satisfy the heads* $\blacklozenge\Box \mathit{fm\_inc} \wedge \blacklozenge\Box \mathit{df} \wedge \blacksquare(\Box \mathit{df} \not\leftrightarrow \mathit{fm\_inc})$ *and* $\blacklozenge\Box \mathit{force\_functional} \wedge \blacklozenge\Box \mathit{force\_physical} \wedge \blacksquare\Box(\mathit{force\_functional} \leftrightarrow \neg \mathit{force\_physical})$*, respectively. There are several transition systems satisfying the conjunct of these two heads:*



Solution B is not desired as it includes only two of four possible combinations of the working hypotheses.

Formally, the independence of two strategies proposing working hypotheses wh_1 and wh_2 means that looking at a state in which wh_1 is active, we cannot derive the truth value of wh_2 in that state.

Definition 7.28 *Independence of Strategies*
Let F be a set of strategies. Let $(S, \rightarrow, t)$ be a consistent transition system wrt. F. The state $s \in S$ satisfies Independence of Strategies*, iff there is no transition system $(S_1, \rightarrow_1, t_1)$ consistent with F such that*

1. *for all working hypotheses $wh_1, wh_2 \in S \cup \neg S$ such that $(S_1, \rightarrow_1, t_1) \models_s \blacksquare\square wh_1 \rightarrow wh_2$ we have $(S, \rightarrow, t) \models_s \blacksquare\square wh_1 \rightarrow wh_2$*
2. *there are working hypotheses $wh_1, wh_2 \in S \cup \neg S$ such that $(S, \rightarrow, t) \models_s \blacksquare\square wh_1 \rightarrow wh_2$ but $(S_1, \rightarrow_1, t_1) \not\models_s \blacksquare\square wh_1 \rightarrow wh_2$*

The transition system $(S, \rightarrow, t)$ satisfies independence of strategies iff every state $s \in S$ satisfies independence of strategies. Treating strategies as independent has several advantages: When writing down strategies we explicitly specify dependencies among certain hypotheses. Independence to other hypotheses need not to be specified. This is important in a case where a strategy formula is added to a large set of existing formulas. Furthermore, assuming independence maximizes our chance to find a solution in the case of non–determinism. If this transition system does not lead to a stable state, there will be no other transition system leading to a consistent state.

In the following we will show that there is only one transition system that satisfies independence of strategies for a given set of strategies. Thus, the semantics is completely specified and can be computed efficiently. In order to combine the transition systems defined by the heads of two strategies while preserving independence, we simply combine the successor states in all possible ways. We call this operation the *State Product*.

Definition 7.29 *State Product*
Given two sets of states S_1 and S_2 the State Product *$S_1 \otimes S_2$ is defined as $\{s_1 \cup s_2 \mid s_1 \in S_1, s_2 \in S_2\}$.*

When constructing the successor transitions for a given state during the diagnostic process, we instantiate the strategies (quantification over components) and collect the heads H_i of the strategies $C_i \rightarrow H_i$, whose conditions C_i are satisfied. We construct the transition systems corresponding to the heads $\{H_i\}$. Then we combine them by applying the following theorem:

Theorem 7.30 *Let $H_1, H_2, \ldots, H_n$ be characteristic formulas of depth* 1 *which have no working hypotheses in common and let $(S_1, \rightarrow_1, t_1), (S_2, \rightarrow_2, t_2), \ldots, (S_n, \rightarrow_n, t_n)$ be the corresponding transition systems. The following transition system $(S, \rightarrow, t)$ satisfies independence of strategies:*

$$S = \emptyset \cup \bigotimes_{i=1}^{n} (S_i \backslash \{\emptyset\}), \quad \rightarrow = \{(\emptyset, s) \mid s \in S\}, \quad t = \emptyset$$

Proof: The proof proceeds in three steps: 1. $(S, \rightarrow, t)$ satisfies the conjunction of all H_i. 2. all transition systems that possibly satisfy the conjunction of all H_i consist of subsets of S. 3. None of these candidates actually satisfies independence of strategies, so $(S, \rightarrow, t)$ in turn does.

Ad 1. In order to show that $(S, \rightarrow, t) \models H_i$ for $1 \leq i \leq n$ it is sufficient to prove that $(S, \rightarrow, t) \models_{\emptyset} \bigwedge_{\emptyset \rightarrow_i s} \blacklozenge F_s \wedge \blacksquare \bigvee_{\emptyset \rightarrow_i s} F_s$, where $F_s = \Box \bigwedge \{wh \mid wh \in s\} \cup \{\neg wh \mid wh \in (\bigcup S_i) \backslash s\}$.

To prove that the two conjuncts hold the definition of the state product is essential. Intuitively, the first conjunct holds as every state in S_i is a subset of at least one state in S. For the second conjunct we argue the other way around: it holds as every state in S is a superset of at least one state in S_i.

Ad 2. As H_i is characteristic and has depth 1 it consists of subformulas $\blacksquare \bigvee_{\emptyset \rightarrow_i s} F_s$. Thus $\bigwedge_{i=1}^{n} H_i$ contains $\bigwedge_{i=1}^{n} \blacksquare \bigvee_{\emptyset \rightarrow_i s} F_s$ which is equivalent to $\blacksquare \bigwedge_{i=1}^{n} \bigvee_{\emptyset \rightarrow_i s} F_s$. As the formulas F_s do not have any working hypotheses in common the latter expression is in minimal CNF. The equivalent DNF corresponds one-to-one to the state product. So the conjunction of H_i can only be satisfied by a subset of the state product.

Ad 3. We show that the first condition for independence of strategies is violated. Let $(S_1, \rightarrow_1, t_1)$ such that $S_1 \subset S$, $\rightarrow_1 \subset \rightarrow$ and $t_1 = t$. Then $(S, \rightarrow, t) \models_s \blacksquare \Box wh_1 \rightarrow wh_2$ for all $wh_1, wh_2 \in S \cup \neg S$ implies $(S_1, \rightarrow_1, t_1) \models_s \blacksquare \Box wh_1 \rightarrow wh_2$ since $S_1 \subset S$. Q.E.D.

The theorem 7.30 describes how to compute the successor states of a given state under the assumption of independent strategies. Iterative application of this theorem yields a straightforward method for computing a transition system satisfying a given set of strategies. In a given state s starting with $\emptyset$ we have to execute the following steps:

1. Compute the diagnoses and corresponding system models under state s.

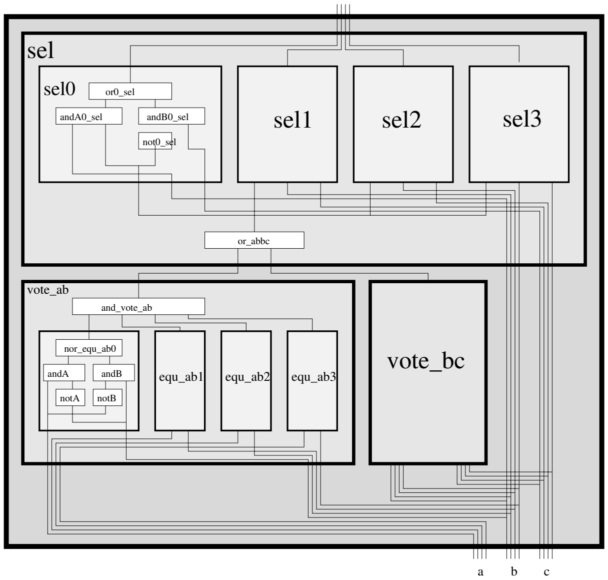


Figure 7.3: Voter

2. Instantiate the body of the strategies using the current diagnoses/models. Collect the heads of the satisfied strategies.

3. Construct a transition system for each head.

4. Combine the resulting transition systems using state product.

The method must be recursively applied to every generated state.

7.6 An Example

A voter (see Figure 7.3) has three 4-bit-inputs a, b, c. It outputs b if $(a = b) \vee (b = c)$ and otherwise c. The equality check is realized by the components *vote_ab* and *vote_bc*. Both are composed of an and-gate and 4 comparators *equ_xy*, which serve as inputs for the and-gate. A comparator *equ_xy* compares 2 bits by realizing the boolean function $x\bar{y} \mathbin{\overline{\vee}} \bar{x}y$ and thus consists of 2 not- and 2 and-gates and a nor-gate. The select component in turn contains 4 one-bit-selectors sel_i which are controlled by the or-gate *or_abbc_sel*. If it is high, selector sel_i lets b_i pass, otherwise c_i. This is realized by 2 and-gates, an or- and a not-gate.

In the process depicted in Figure 7.4 the three input words are all 0000 and the output is observed to be 1111. The top level diagnosis uniquely identifies *sel* as abnormal (1), but the following refinement does not lead to any diagnoses (2). So the single-fault-assumption is relaxed and two successor states are created, allowing double faults and incomplete fault models, respectively. With incomplete fault modes

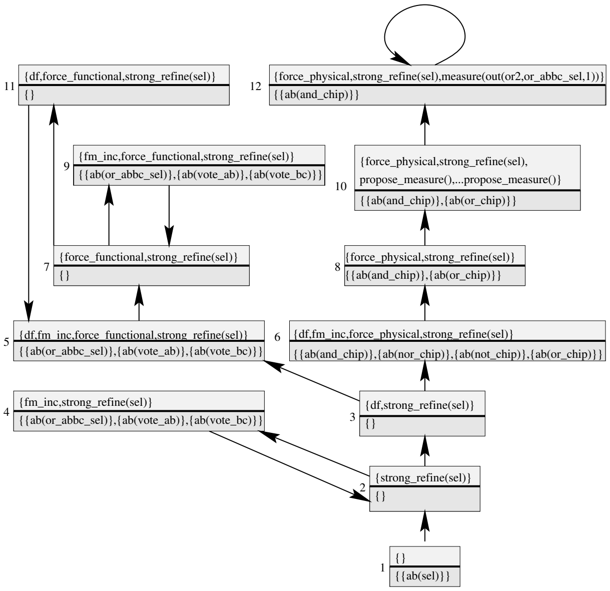

11
{df,force_functional,strong_refine(sel)}
{}
12
{force_physical,strong_refine(sel),measure(out(or2,or_abbc_sel,1))}
{{ab(and_chip)}}
9
{fm_inc,force_functional,strong_refine(sel)}
{{ab(or_abbc_sel)},{ab(vote_ab)},{ab(vote_bc)}}
10
{force_physical,strong_refine(sel),
propose_measure(),...propose_measure()}
{{ab(and_chip)},{ab(or_chip)}}
7
{force_functional,strong_refine(sel)}
{}
8
{force_physical,strong_refine(sel)}
{{ab(and_chip)},{ab(or_chip)}}
5
{df,fm_inc,force_functional,strong_refine(sel)}
{{ab(or_abbc_sel)},{ab(vote_ab)},{ab(vote_bc)}}
6
{df,fm_inc,force_physical,strong_refine(sel)}
{{ab(and_chip)},{ab(nor_chip)},{ab(not_chip)},{ab(or_chip)}}
4
{fm_inc,strong_refine(sel)}
{{ab(or_abbc_sel)},{ab(vote_ab)},{ab(vote_bc)}}
3
{df,strong_refine(sel)}
{}
2
{strong_refine(sel)}
{}
1
{}
{{ab(sel)}}

Figure 7.4: A diagnostic process.

some diagnoses are found (4). Since the hypothesis of incomplete fault modes is not monotonic, we have to drop it again. The consequence is a loop between two states, in which only the state with incomplete fault modes is consistent. By definition 7.19 we have reached a weakly stable state. The search for double faults (3) in the other branch is not successful. Two strategies apply in this situation. In all successor states we have to allow incompleteness of fault models in addition to double faults (section 7.4). Furthermore, we have to branch between physical and functional view as proposed by the multiple views strategy (section 7.4). Two successor states are generated:

- With double faults and incomplete fault modes three diagnoses are found (5). The search for more preferred diagnoses first leads to no diagnoses (7). Allowing double faults does not help (11), while dropping the completeness of fault modes assumption yields three single faults (9), so that this state is again weakly stable.

- Beside the computations in the functional view, we obtain diagnoses of the physical view (6). With double faults and incomplete fault modes allowed, five diagnoses are consistent with the observation. Thus, in the next step the preferences are relaxed and *and_chip* and *or_chip* are valid diagnoses in the physical view (8). In order to discriminate among those two diagnoses several measurements are proposed (10). Among them the point *or_abbc_sel* is chosen and finally the state $\{force\_physical, strong\_refine(sel), measure(out(or2, or\_abbc\_sel, 1))\}$ is stable.

7.7 Relation to other Formalisms

Let us reconsider the way we use modal logic in our operational semantics. Starting from a given state with no working hypotheses active we construct the S5-worlds for this state. These worlds are given by the diagnosis semantics. Based on the truth values of the formulas in this world a specialized algorithm computes transitions (with certain properties) to the following process states. By iterating this algorithm we obtain a transition system. Thus, besides providing a formal framework we use modal logic mainly for model generation. In this sense, our work is more motivated by dynamic logics [Eme90] where model checking and generation are main issues than by classical modal logic.

Furthermore, modal deduction [ndCH95] is computationally very complex. Diagnosis strategies are means of speeding up diagnosis by proposing suitable assumptions. So, performing modal deduction during the diagnosis process would slow down computation. Our model generation algorithm is restricted to the given problem because it exploits the problem structure and therefore is quite efficient. A modal deduction system might still be useful here, it could be used to check if a given set of strategies is consistent, independent of a diagnostic problem. However, such a system is currently outside the scope of this work.

Another possible formalism for our framework is metaprogramming. Metaprogramming is suitable for implementation rather than for providing a formal background. In fact, a former version [DNPS95] of our formalization and the current one have been implemented on top of the non-monotonic reasoning system REVISE using meta–logic programming with a *demo*-predicate for both, provability under well-founded semantics and abduction of conflicts [DNPS95].

7.8 Discussion

To cope with large-scale systems the theory of model-based diagnosis has been extended to include concepts such as multiple views [Dav84, Ham91], hierarchies [Dav84, Ham91, Gen84, Moz91, BD94], preferences [DPN94, FNS94] and measurements [dKRS91]. Struss introduced the idea of diagnosis as process [Str92], further developed by Böttcher and Dressler [BD93, BD94]. We formalized it by defining a meta-language that allows to describe the process declaratively. In this chapter we focused on two important issues. First, we showed how to design strategies to cover the concepts mentioned above. Second, we developed an operational semantics and an algorithm that processes these strategies and efficiently computes the diagnostic process. We identified generic one-step strategies to deal with monotonic and non-monotonic working hypotheses as well as deterministic and non-deterministic strategies. In particular, the combination of non-monotonic working hypotheses and non-determinism allowed us to express preferences which usually have to be treated in a separate framework [DPN94, FNS94]. We showed how to use multiple views and how to employ hierarchies by the strategy of structural refinement. We integrated measurement strategies using procedural attachment. Beside the practical motivation of one-step strategies we proved that one-step strategies are universal, i.e. every diagnostic process can be fully characterized by a set of one-step strategies. Furthermore, we defined characteristic formulas and independence of strategies which lead to an efficient algorithm that covers the whole variety of strategies. The design and evaluation of these strategies was evaluated in the domain of digital circuits using the voter circuit from [Isc85].

Chapter 8

Conclusion

In this thesis we have described an efficient engine for model–based diagnosis, based on a very general framework for non–monotonic reasoning. We have evaluated this engine using two technical diagnosis applications: The diagnosis of combinatorial circuits and the diagnosis of cellular networks. We have complemented our efficient algorithms with a declarative modeling language for the diagnosis process.

8.1 Contributions

We have reviewed several diagnostic concepts from the model–based diagnosis literature to find a suitable diagnosis definition. Because of its flexibility, we chose spectrum diagnosis, i.e. the diagnosis definition by Console and Torasso [CT91], complemented by different minimality concepts. Spectrum diagnoses consist of a consistency–based and an abductive part. We critically examined the claim made by Console and Torasso, that abductive diagnoses can be replaced by consistency–based diagnosis. Our result is that their statement does not hold for systems with a network structure, which are ubiquitous in technical diagnosis. Thus, we found that a flexible diagnoser must implement both consistency–based and abductive diagnosis.

Instead of a direct implementation of the diagnostic concepts using dedicated conflict–based algorithms, we chose to embed diagnosis in a general framework for non–monotonic reasoning, thereby increasing the expressiveness and flexibility of our implementation. By reducing consistency–based and abductive diagnosis to circumscription, we established a different view on the relationship between diagnosis and circumscription. Instead of using circumscription to define the semantics of new diagnostic concepts [BC94, Rai90], we use circumscription to reconstruct and efficiently implement widely used diagnostic definitions.

The basis of our framework for non–monotonic reasoning is a novel implementation of circumscription for fixed domain theories. In contrast to previous implementations, which focused on a restricted variant of circumscription (like [Gin89]) or demanded certain structural properties of the underlying theories (like [Lif85]) our algorithms handle prioritized and parallel circumscription, using both varying and fixed predicates on all fixed domain theories.

Two technical diagnosis applications have been solved using the resulting DRUM–II system: The diagnosis of combinatorial circuits and the alarm correlation in cellular networks.

Due to their size and complicated internal structure, combinatorial circuits are regarded as a good benchmark for model–based diagnosis engines. The study of the redundant computation steps of DRUM–II on this application has motivated the extension of the basic DRUM–II algorithm by an additional focusing technique. The optimized DRUM–II algorithm exploits precompiled information on the structure of the device under consideration. The size of this precompiled information is quadratic in the size of the system description. Since the optimized DRUM–II algorithm does not record any further information during the search for models, its memory requirements are more predictable than those of most ATMS–based reasoners, whose data structures exhibit combinatorial explosion during search. Our experiments have shown that the optimized DRUM–II system is much more efficient than all previous systems in nearly all cases on the ISCAS–85 benchmark suite of combinatorial circuits.

While the diagnosis of combinatorial circuits mainly posed an efficiency problem, the alarm correlation in cellular networks was a challenging modeling problem. Our solution is to our knowledge the first fully model–based approach to alarm correlation in cellular networks. Its advantages are: robustness against topology reconfigurations, good performance on noisy data, as well as the ability to diagnose multiple faults. Despite these advantages our consistency–based system description was restricted to tree–structured networks. Therefore we have developed a similar system description based on spectrum diagnosis, which shares the advantages of the consistency–based description, but is furthermore correct for any topology. Thus, in this application the greater flexibility of the DRUM–II inference mechanism compared to previous engines has enabled us to provide a more general system description.

In many applications of DRUM and DRUM–II it has proved useful to compute diagnoses in two steps: First, a model of the correct behavior given the input values of the system is computed. Then, this model is updated with the observed symptoms. This incremental procedure is one of the key concepts in DRUM–II. We have shown that this idea can be successfully integrated in reasoners based on different technology. The integration of our two step approach into a theorem prover has made the resulting system (NIHIL) capable of solving diagnosis problems with reasonable efficiency.

We have complemented our efficient diagnosis algorithms in DRUM–II with a declarative language and an efficient algorithm for exploiting diagnosis strategies. Diagnosis strategies, which characterize the diagnostic process are usually encoded as part of the diagnosis engine. Our bimodal language for strategies allows to describe the process in the same declarative fashion as the system description describes the device under consideration.

8.2 Future Work

The work presented in this thesis should be extended in two main directions: First, the DRUM–II engine should be integrated in an autonomous logical agent, which is able to solve diagnosis tasks in a distributed environment. Second, further applications of DRUM–II should be investigated.

We have started addressing the issues of model–based diagnosis in a distributed environment in [FN96c, FdAMNS97]. In the environment of a spatially distributed system, agents must communicate with each other to find a globally consistent system diagnosis. Furthermore, an autonomous diagnosis agent should be capable of handling monitoring and control tasks. To meet these requirements, DRUM–II can be extended by perception and action mechanisms. To react to changes in the environment in real time, a reactive rule mechanism is needed.

The extension of DRUM–II to an autonomous agent would permit to use DRUM–II as a distributed diagnosis agent for telecommunication networks, communicating with other agents according to the ITU TMN standard. Introducing distributed diagnosis agents in large telecommunication networks could reduce the message traffic in case of errors.

Another interesting application area for DRUM–II is the diagnosis of bridge faults in digital circuits. Bridge faults are very common in modern CMOS circuits [CLFL95]. They can be handled through explicit modeling [Böt95]. Another interesting approach to bridge fault diagnosis [CLFL95, LLC96] is the reduction of bridge faults to combinations of stuck–at faults. It is an interesting future application to solve these complex, multiple fault scenarios with DRUM–II.

Finally, the temporal reasoning capabilities of DRUM–II can be exploited in temporal diagnosis applications. Within the telecommunication domain the detection and diagnosis of trends concerning network traffic or the bit error rate of microwave links are promising application areas.

Bibliography

[Bak91] A. B. Baker. Nonmonotonic reasoning in the framework of the situation calculus. *Artificial Intelligence*, 49:5–23, 1991.

[BC94] Philippe Besnard and Marie-Odile Cordier. Explanatory diagnoses and their characterization by circumscription. *Annals of Mathematics and Artificial Intelligence*, 11:75–96, 1994.

[BD93] Claudia Böttcher and Oskar Dressler. Diagnosis process dynamics: Holding the diagnostic trackhound in leash. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 2, pages 1460–1471. Morgan Kaufmann Publishers, Inc., 1993.

[BD94] Claudia Böttcher and Oskar Dressler. A framework for controlling model-based diagnosis systems with multiple actions. *Annals of Mathematics and Artificial Intelligence, special Issue on Model-based Diagnosis*, 11(1–4), 1994.

[Bea93] Simona Brugnosi and Roberto Manione et al. An expert system for real time fault diagnosis of the italian telecommunications network. In *Integrated Network Management III*. Elsevier North Holland, 1993.

[BFFN97a] Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, and Wolfgang Nejdl. Tableaux for diagnosis applications. In *International Conference on Analytic Tableaux and Related Methods (Springer LNAI 1227)*, Pont-a-Mousson, May 1997.

[BFFN97b] Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, and Wolfgang Nejdl. Semantically guided theorem proving for diagnostic applications. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, August 1997.

[BFN96] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper tableaux. In *European Workshop on Logic in AI (JELIA 96), LNAI 1126*. Springer, 1996.

[BML89] I. Bratko, I. Mozetič, and N. Lavrač. *KARDIO: A Study in Deep and Qualitative Knowledge for Expert Systems*. MIT Press, Cambridge, MA, 1989.

[BNSS93] Thomas Brunner, Wolfgang Nejdl, Harald Schwarzjirg, and Monika Sturm. Online expert system for power system diagnosis and restoration. *Intelligent Systems Engineering*, pages 15–24, Spring 1993.

[Böt95] Claudia Böttcher. No Faults in Structure? - How to Diagnose Hidden Interactions. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1728–1734, Montréal, Canada, August 1995. Morgan Kaufmann Publishers, Inc.

[BS84] Bruce G. Buchanan and Edward H. Shortliffe, editors. *Rule-Based Expert System - The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley Publishing Company, 1984.

[BY96] F. Bry and A. Yahya. Minimal Model Generation with Positive Unit Hyper–Resolution Tableaux. In P. Moscato, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods, LNAI 1071*, pages 143–159. Springer, 1996.

[CDT91] Luca Console, Daniele Theseider Dupré, and Pietro Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[CL73] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[CLFL95] B. Chess, D. B. Lavo, F. J. Ferguson, and T. Larrabee. Diagnosis of Realistic Bridging Faults with Single Stuck–at Information. In *Proceedings of the International Conference on Computer-Aided Design*, pages 185–192, 1995.

[CP94] H. Chu and D. Plaisted. Semantically Guided First–Order Theorem Proving using Hyper–Linking. In *Automated Deduction - CADE 12, LNAI 814*, pages 192–206, Nancy, France, June 1994. Springer.

[CT91] Luca Console and Pietro Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3):133–141, 1991. Also in [HCd92].

[CW91] Timothy S-C Chou and Marianne Winslett. Immortal: A model-based belief revision system. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 99–110, Cambridge, April 1991. Morgan Kaufmann Publishers, Inc.

[CW94] T. S-C. Chou and M. Winslett. A model–based belief revision system. *Journal of Automated Reasoning*, 12:157–208, 1994.

[Dav84] Randall Davis. Diagnostic reasoning based on structure and behaviour. *Artificial Intelligence*, 24:347–410, 1984.

[DH88] Randall Davis and Walter Hamscher. Model-based reasoning: Troubleshooting. In *Exploring Artificial Intelligence*, chapter 8, pages 297–346. Morgan Kaufmann Publishers, Inc., 1988.

[dK76] Johan de Kleer. Local methods for localizing faults in electronic circuits. Technical Report AI Memo 394, MIT, Cambridge, MA, 1976.

[dK90a] J. de Kleer. Eliminating the fixed predicates from a circumscription. *Artificial Intelligence*, 39(3):391–398, 1990.

[dK90b] Johan de Kleer. Using crude probability estimates to guide diagnosis. *Artificial Intelligence*, 45:381–391, 1990.

[dK91] Johan de Kleer. Focusing on probable diagnoses. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 842–848, Anaheim, July 1991. Morgan Kaufmann Publishers, Inc.

[dKRS91] Johan de Kleer, Olivier Raiman, and Mark Shirley. One step lookahead is pretty good. In *Second International Workshop on the Principles of Diagnosis*, Milano, Italy, October 1991.

[dKW87] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[dKW89] Johan de Kleer and Brian C. Williams. Diagnosis with behavioral modes. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1324–1330, Detroit, August 1989. Morgan Kaufmann Publishers, Inc.

[dMR92] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2–3):197–222, 1992. Also in [HCd92].

[DNPS95] C. V. Damásio, W. Nejdl, L. Pereira, and M. Schroeder. Model-based diagnosis preferences and strategies representation with meta logic programming. In Krzysztof R. Apt and Franco Turini, editors, *Meta-logics and Logic Programming*, chapter 11, pages 269–311. The MIT Press, 1995.

[Doh94] P. Doherty. Reasoning about action and change using occlusion. In *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 401–405, 1994.

[Dow92] Keith L. Downing. Consistency-based diagnosis in physiological domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 558–563, San Jose, California, July 1992.

[Dow93] Keith L. Downing. Physiological applications of consistency-based diagnosis. *Artificial Intelligence in Medicine*, 5(1):9–30, 1993.

[DPN94] Carlos Viegas Damásio, Luís Moniz Pereira, and Wolfgang Nejdl. Revise: An extended logic programming system for revising knowledge bases. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 607–618, Bonn, Germany, May 1994. Morgan Kaufmann Publishers, Inc.

[DS92] O. Dressler and P. Struss. Back to defaults: Characterizing and computing diagnoses as coherent assumption sets. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 719–723, 1992.

[dS95] Luis da Silva. Final report on gema demonstrator. Technical report, CET/Portugal Telecom, Aveiro, Portugal, 1995.

[Eme90] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16. Elsevier, Amsterdam, New York, 1990.

[FdAMNS97] Peter Fröhlich, Iara de Almeida Móra, Wolfgang Nejdl, and Michael Schroeder. Diagnostic agents for distributed systems. In *4th ModelAge Workshop – Formal Models Of Agents*, 1997.

[FN96a] Peter Fröhlich and Wolfgang Nejdl. A model–based reasoning approach to circumscription. In *Proceedings of the 12th European Conference on Artificial Intelligence*, 1996.

[FN96b] Peter Fröhlich and Wolfgang Nejdl. A model–based reasoning approach to circumscription – extended version. In *Sixth International Workshop on Nonmonotonic Reasoning*, 1996.

[FN96c] Peter Fröhlich and Wolfgang Nejdl. Resolving conflicts in distributed diagnosis. In *ECAI Workshop on Modelling Conflicts in AI*, 1996.

[FN97] Peter Fröhlich and Wolfgang Nejdl. A static model-based engine for model-based reasoning. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, August 1997.

[FNJW97] Peter Fröhlich, Wolfgang Nejdl, Klaus Jobmann, and Hermann Wietgrefe. Model-based alarm correlation in cellular phone networks. In *Fifth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, January 1997.

[FNS] Peter Fröhlich, Wolfgang Nejdl, and Michael Schroeder. Using modal logic to define strategies in model-based diagnosis. Workshop on Model Theory and Complexity, Sevilla, 1995.

[FNS94] Peter Fröhlich, Wolfgang Nejdl, and Michael Schröder. A formal semantics for preferences and strategies in model-based diagnosis. In *5th International Workshop on Principles of Diagnosis (DX-94)*, pages 106–113, New Paltz, NY, October 1994.

[FNS96] Peter Fröhlich, Wolfgang Nejdl, and Michael Schroeder. Design and implementation of diagnostic strategies using modal logic. In *JELIA'96 – European Workshop on Logic in AI (LNAI 1126)*. Springer-Verlag, 1996.

[FNS98] Peter Fröhlich, Wolfgang Nejdl, and Michael Schroeder. Strategies in model-based diagnosis. *Journal of Automated Reasoning*, 20(1 and 2):81–105, April 1998.

[FSW95] G. Friedrich, M. Stumptner, and F. Wotawa. Model–Based Diagnosis of Hardware Designs. In W. Nejdl, editor, *Sixth International Workshop on Principles of Diagnosis (DX–95)*, Goslar, Germany, 1995.

[Gen84] M. R. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436, 1984.

[Gin89] M. L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

[GMW96] H. Ganzinger, C. Meyer, and C. Weidenbach. Soft typing for ordered resolution. Unpublished, 1996.

[GN97] Johann Gamper and Wolfgang Nejdl. Abstract temporal diagnosis in medical domains. *Artificial Intelligence in Medicine*, 1997. to appear.

[GSW89] 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, 1989.

[Ham91] Walter C. Hamscher. Modeling digital circuits for troubleshooting. *Artificial Intelligence*, 51(1-3):223–271, October 1991.

[HCd92] W. Hamscher, L. Console, and J. de Kleer. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.

[Isc85] The ISCAS-85 Benchmarks. http://www.cbl.ncsu.edu/www/CBL_Docs/iscas85.html, 1985.

[JW93] Gabriel Jakobson and Mark D. Weissmann. Alarm correlation. *IEEE Network*, November 1993.

[Kar94] G. N. Kartha. Two counterexamples related to baker's approach to the frame problem. *Artificial Intelligence*, 69:379–391, 1994.

[KNH91] Walter Kehl, Mark Newstead, and Heiner Hopfmüller. A model-based reasoning system for the maintenance of telecommunication networks. In *Proceedings of the International Workshop on Expert Systems and Their Applications*, May 1991.

[KS96] H. Kautz and B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Portland, Oregon, 1996. AAAI Press.

[Lar92] T. Larrabee. Test Pattern Generation Using Boolean Satisfiability. *IEEE Transactions on Computer–Aided Design*, pages 4–15, jan 1992.

[Lif85] V. Lifschitz. Computing circumscription. In *Proc. of the Intl. Conf. on Artificial Intelligence*, pages 121–127, Los Angeles, August 1985.

[Lif86] V. Lifschitz. On the Satisfiability of Circumscription. *Artificial Intelligence*, 28:17–27, 1986.

[LLC96] D.B. Lavo, T. Larrabee, and B. Chess. Beyond the Byzantine Generals: Unexpected Behaviour and Bridging Fault Diagnosis. In *Proceedings of the International Test Conference*, pages 611–619, October 1996.

[Luk90] W. Lukaszewicz. *Non–monotonic reasoning: formalization of commonsense reasoning*. Ellis Horwood, 1990.

[MB88] Rainer Manthey and Francois Bry. Satchmo: A theorem prover implemented in Prolog. In *Proceedings of the International Conference on Automated Deduction*, Argonne, Illinois, May 1988. Springer-Verlag.

[McC80] J. McCarthy. Circumscription - a form of non–monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[McC86] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–118, 1986.

[Moz91] Igor Mozetič. Hierarchical model-based diagnosis. *International Journal of Man-Machine Studies*, 35:329–362, 1991.

[MT95] Marita Möller and Stefan Tretter. Event correlation in network management systems. In *Proceedings of the 15th International Switching Symposium*, volume 2, Berlin, 1995.

[ndCH95] Luis Fari nas dec Cerro and Andreas Herzig. Modal deduction with applications in epistemic and temporal logics. In C.J. Hogger Dov M. Gabbay and J.A. Robinson, editors, *The Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, pages 499–594. Oxford Science Publications, 1995.

[Ned93] Bernhard Nedizavec. Implementierung von Belief Revision. Master's thesis, TU Wien, 1993. Prof. Wolfgang Nejdl.

[NF96] Wolfgang Nejdl and Peter Fröhlich. Minimal model semantics for diagnosis – techniques and first benchmarks. In *Seventh International Workshop on Principles of Diagnosis*, Val Morin, Canada, October 1996.

[NFS95] Wolfgang Nejdl, Peter Fröhlich, and Michael Schroeder. A formal framework for representing diagnosis strategies in model–based diagnosis systems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1721–1727, Montréal, Canada, August 1995. Morgan Kaufmann Publishers, Inc.

[NG94] Wolfgang Nejdl and Brigitte Giefer. DRUM:Reasoning without conflicts and justifications. Technical report, RWTH Aachen, May 1994. Submitted for publication.

[Out93] Dirk-Jan Out. *Strategies for Efficient Model–Based Troubleshooting*. PhD thesis, Universiteit Twente, 1993.

[PN93] Monika Pfau and Wolfgang Nejdl. Integrating model-based and heuristic features in a realtime expert system for power distribution networks. *IEEE Expert*, pages 12–18, August 1993.

[Pop82] H. E. Pople. Heuristic Methods for Imposing Structure in Ill–Structured Problems: The Structuring of Medical Diagnosis. In P. Szolovits, editor, *Artificial Intelligence in Medicine*, Boulder, CO, 1982. Westview Press.

[Prz89] Teodor C. Przymusinki. An algorithm to compute circumscription. *Artificial Intelligence*, 38:49–73, 1989.

[Rai90] Olivier Raiman. Circumscribing diagnosis engines: Exploiting the alibi principle to avoid blind search. In *Proceedings of the International Workshop on Expert Systems in Engineering*, Vienna, September 1990. Springer-Verlag. Lecture Notes in AI.

[RC81] R. Reiter and G. Criscuolo. On interacting defaults. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 270–276, 1981.

[RdKS93] Olivier Raiman, Johan de Kleer, and Vijay Saraswat. Critical reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 18–23, Chambery, August 1993.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1–2), 1980.

[Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[Rob65] J. A. Robinson. Automated deduction with hyper–resolution. *Internat. J. Comput. Math.*, 1:227–234, 1965.

[San94] E. Sandewall. *Features and Fluents. The representation of Knowledge about Dynamical Systems. Volume I.* Oxford University Press, 1994.

[SD89] Peter Struss and Oskar Dressler. Physical negation — Integrating fault models into the general diagnostic engine. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1318–1323, Detroit, August 1989. Morgan Kaufmann Publishers, Inc.

[SD96] Barbara M. Smith and Martin E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:155–181, 1996.

[SMS95] P. Struss, A. Malik, and M. Sachenbacher. Qualitative Modeling is the Key. In W. Nejdl, editor, *Sixth International Workshop on Principles of Diagnosis (DX–95)*, Goslar, Germany, 1995.

[SPBL95] Kelvyn Scrupps, Dominic Pang, John Bigham, and Mike Laughton. Final report on gema evaluation and enhancement of gms tools and technology. Technical report, Queen Mary and Westfield College, London, UK, 1995.

[Str92] Peter Struss. Diagnosis as a process. In W. Hamscher, L. Console, and J. de Kleer, editors, *Readings in Model-Based Diagnosis*, pages 408–418. Morgan Kaufmann Publishers, Inc., 1992.

[Win88] Marianne Winslett. Reasoning about action using a possible models approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 89–93, Saint Paul, Minnesota, August 1988.

[WKA78] S. M. Weiss, C. A. Kulikowski, and S. Amarel. A Model–Based Method for Computer-Aided Medical Decision–Making. *Artificial Intelligence*, 11:145–172, 1978.

[WN96a] B. C. Williams and P. P. Nayak. Immobile Robots: AI in the New Millennium. *AI Magazine*, 1996. Fall Issue.

[WN96b] Brian C. Williams and Pandurang Nayak. A model–based approach to reactive self–configuring systems. In *Thirteenth National Conference on Artificial Intelligence (AAAI–96)*, Portland, Oregon, 1996.

[WN97] Brian C. Williams and Pandurang Nayak. A model–based approach to reactive self–configuring systems. In *7th International Workshop on Principles of Diagnosis*, Val Morin, Canada, 1997.

[WS97] T. Wakaki and K. Satoh. Compiling Prioritized Circumscription into Extended Logic Programs. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, August 1997.

[WTJ$^+$97] Hermann Wietgrefe, Klaus-Dieter Tuchs, Klaus Jobmann, Guido Carls, Peter Fröhlich, Wolfgang Nejdl, and Sebastian Steinfeld. Using neural networks for alarm correlation in cellular phone networks. In *Proceedings of the International Workshop on Applications of Neural Networks in Telecommunications*, 1997.

[Yea96] Shaula Yemini and Shmuel Klinger et al. High speed and robust event correlation. *IEEE Communications Magazine*, May 1996.

Lebenslauf

Name: Peter Fröhlich

Geburtsdatum: 12.2.1970

Schulausbildung

1976–1980 Kath. Grundschule Würselen

1980–1989 Heilig–Geist–Gymnasium Würselen

am 18.5.1989 Abitur

Wehrdienst

1989–1990 Zivildienst in Würselen

Studium

1990–1995 Studium der Informatik mit Nebenfach Wirtschaftswissenschaften an der RWTH Aachen

Diplomarbeit zum Thema *Modellierung des Diagnoseprozesses in modellbasierten Systemen*, Note: Sehr Gut

am 4.7.1995 Diplom mit der Gesamtnote: Mit Auszeichnung

Beruflicher Werdegang

1991–1994 Tätigkeit als freier Mitarbeiter bei debis Systemhaus GEI in Aachen

seit 1995 Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Rechnergestützte Wissensverarbeitung der Universität Hannover

Anfertigung der vorliegenden Dissertation unter Betreuung von Prof. Nejdl