Tableaux for Diagnosis Applications

Peter Baumgartner, Peter Fröhlich,
Ulrich Furbach, Wolfgang Nejdl

23/96

Tableaux for Diagnosis Applications

Peter Baumgartner[1], Peter Fröhlich[2], Ulrich Furbach[1], Wolfgang Nejdl[2]

[1] Univ. Koblenz, Inst. f. Informatik
E-mail: {peter,uli}@informatik.uni-koblenz.de
[2] Universität Hannover
E-mail: {froehlich,nejdl}@kbs.uni-hannover.de

Abstract. In [Nejdl and Fröhlich, 1996] a very efficient system for solving diagnosis tasks has been described, which is based on belief revision procedures and uses first order logic system descriptions. In this paper we demonstrate how such a system can be rigorously formalized from the viewpoint of deduction by using the calculus of hyper tableaux [Baumgartner *et al.*, 1996]. The benefits of this approach are twofold: first, it gives us a clear logical description of the diagnosis task to be solved; second, as our experiments show, the approach is feasible in practice and thus serves as an example of a successful application of deduction techniques to real-world applications.

1 Introduction

In this paper we will demonstrate that model generation theorem proving is very well suited for solving consistency-based diagnosis tasks. According to Reiter ([Reiter, 1987]) a simulation model of the technical device under consideration is constructed and is used to predict its normal behavior. By comparing this prediction with the actual behavior it is possible to derive a diagnosis. We will use a proof procedure, which is an implementation of the hyper tableaux calculus presented in [Baumgartner *et al.*, 1996].

This work was motivated by the study of the diagnosis system DRUM-2 [Fröhlich and Nejdl, 1996, Nejdl and Fröhlich, 1996]. This system is in essence a model generation procedure which takes into account the particularities of logical descriptions of a diagnosis task. The basic idea is to start with an initial model of a correctly functioning device. This model is revised, whenever the actual observations differ from the predicted behavior. We adapt this idea to our hyper tableaux calculus, which yields *semantic hyper tableaux*. The resulting system is comparable in efficiency to the DRUM-2 system. We know of no other general purpose theorem prover which has been used to solve large diagnosis problems.

In the following section we formally describe the diagnosis task. We then show how hyper tableaux can be used to compute diagnoses. In section 4 we describe the strategy of DRUM-2 and in section 5 we adapt this refinement coming up with semantic hyper tableaux. Finally we describe the implementation and its results.

We assume that the reader is familiar with the basic concepts of propositional logic. *Clauses*, i.e. multisets of literals, are usually written as the disjunction $A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n$ or as an implication $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ $(m \geq 0,\ n \geq 0)$. With $\overline{L}$ we denote the complement of a literal L. Two literals L and K are *complementary* if $\overline{L} = K$.

2 The Diagnosis Task

2.1 Model–Based Diagnosis

Heuristic rule–based expert systems were the first approach to automated diagnosis. The knowledge bases of such systems could not be easily modified or verified to be correct and complete. These difficulties have been overcome by the introduction of model–based diagnosis [Reiter, 1987], where a simulation model of the device under consideration is used to predict its normal behavior, given the observed input parameters. Diagnoses are computed by comparison of predicted vs. actual behavior (Fig. 1).

This approach uses an extendible logical description of the device, called the system description (SD), formalized by a set of first–order formulas. The system description consists of a set of axioms characterizing the behavior of system components of certain types. The topology is modeled separately by a set of facts. We will now define the diagnostic concept mathematically. The diagnostic problem is described by system description SD, a set $COMP$ of components and a set OBS of observations (logical facts). With each component we associate a behavioral mode: $Mode(c, Ok)$ means that component c is behaving correctly, while $Mode(c, Ab)$ (abbreviated by $Ab(c)$) denotes that c is faulty.

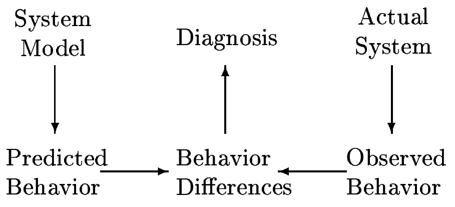

Fig. 1. Model–Artifact Difference

A *Diagnosis* D is a set of faulty components, such that the observed behavior is consistent with the assumption, that exactly the components in D are behaving abnormally. If a diagnosis contains no proper subset which is itself a diagnosis, we call it a *Minimal Diagnosis*.

Definition 1 (Reiter 87). A *Diagnosis* of $(SD, COMP, OBS)$ is a set $\Delta \subseteq COMP$, such that

$$SD \cup OBS \cup \{Mode(c, Ab) | c \in COMP\} \cup \{\neg Mode(c, Ab) | c \in COMP - \Delta\}$$

is consistent. Δ is called a *Minimal Diagnosis*, iff it is the minimal set (wrt. $\subseteq$) with this property.

Minimal Diagnoses are a natural concept, because we do not want to assume that a component is faulty, unless this is necessary to explain the observed behavior. Diagnosis on the basis of definition 1 is today mostly referred to as *Consistency Based Diagnosis* because it postulates that the diagnoses are logically consistent with the system description and the observations. See [Console and Torasso, 1991] for an overview of alternative diagnostic definitions. To compute diagnoses of $(SD, COMP, OBS)$ it is sufficient to compute models of $SD \cup OBS$.

Corollary 2. *Let M be a model of $SD \cup OBS$. Then $M|Ab|$ is a diagnosis of $(SD, COMP, OBS)$.*

2.2 Restricting the Cardinality of Diagnoses

The set of all minimal diagnoses can be large for complex technical devices. Therefore, stronger criteria than minimality are often used to further discriminate among the minimal diagnoses. These criteria are usually based on the probability or cardinality of diagnoses. In the remainder of this paper we will use restrictions on the cardinality of diagnoses. We can restrict the cardinality of each diagnosis Δ to at most n abnormals (i.e. $|\Delta| \leq n$) by adding the so called *n–Fault–Assumption* to the system description.

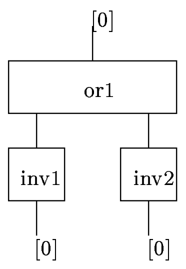

Fig. 2. A Circuit

Definition 3. The sentence

$$N\_Faults \leftrightarrow (\forall_{i=1}^{n+1} x_i : \bigwedge_{i=1}^{n+1} Ab(x_i) \rightarrow \bigvee_{i,j=1,\, i \neq j}^{n+1} x_i = x_j)$$

is called the *n–Fault–Assumption*.

A widely used example of the n–Fault–Assumption is the 1–Fault–Assumption or *Single Fault Assumption*. Many specialized systems for technical diagnosis have the Single Fault Assumption implicit and are unable to handle multiple faults. In model–based diagnosis systems the Single Fault Assumption can be activated explicitly in order to provide more discrimination among diagnoses and to speed up the diagnosis process.

Example 4. As a running example for this paper consider the simple digital circuit in figure 2.2 consisting of an or–gate (*or1*) and two inverters (*inv1* and *inv2*). Its function can be described by the following propositional formulas[1].

$$\begin{aligned}
\mathit{OR1} &: \neg(ab(or1)) \rightarrow high(or1,o) \leftrightarrow (high(or1,i1) \vee high(or1,i2))\\
\mathit{INV1} &: \neg(ab(inv1)) \rightarrow high(inv1,o) \leftrightarrow \neg(high(inv1,i))\\
\mathit{INV2} &: \neg(ab(inv2)) \rightarrow high(inv2,o) \leftrightarrow \neg(high(inv2,i))\\
\mathit{CONN1} &: high(inv1,o) \leftrightarrow high(or1,i1)\\
\mathit{CONN2} &: high(inv2,o) \leftrightarrow high(or1,i2)
\end{aligned}$$

Thus we have $SD = \{OR1, INV1, INV2, CONN1, CONN2\}$ as the system description. We observe that both inputs of the circuit have low voltage and the output also has low voltage, i.e.

$$OBS = \{LOW\_INV1\_I, LOW\_INV1\_I, LOW\_OR1\_O\} \ .$$

Or, as formulas:

$$\begin{aligned}
LOW\_INV1\_I &: \neg(high(inv1,i))\\
LOW\_INV1\_I &: \neg(high(inv2,i))\\
LOW\_OR1\_O &: \neg(high(or1,o))
\end{aligned}$$

The expected behavior of the circuit given that both inputs are low would be high voltage at the outputs of both inverters and consequently high voltage at the output of the or–gate. This model of the correctly functioning device

$$I_0 = \{high(inv1,o), high(inv2,o), high(or1,i1), high(or1,i2), high(or1,o\}$$

can be computed very efficiently even for large devices by domain–specific tools, e.g. circuit simulators.

In the next section we will review the hyper tableaux calculus and we will show how it can be used to generate models for a diagnosis task.

[1] These formulas can be obtained by instantiating a first order description of the gate functions with the structural information.

3 Hyper Tableaux Calculus

In [Baumgartner *et al.*, 1996] we 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 [Baumgartner *et al.*, 1996] for a detailed discussion. In order to make the present paper 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 $\mathcal{S}$ 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 often will identify nodes by their labels.

A *branch* of a tableau T is a sequence $N_0, \ldots, N_n$ $(n \geq 0)$ of 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 $\mathrm{lit}(b) = \{N_1, \ldots N_n\}$. 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 $A \in \mathrm{lit}(b)$. 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 5 (Hyper tableau). A literal set is called *inconsistent* iff it contains a pair of complementary literals, otherwise it is called *consistent*. *Hyper tableaux* for $\mathcal{S}$ and 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. $C = A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ is a clause from $\mathcal{S}$ ($m \geq 0$, $n \geq 0$), called *extending clause* in this context, and
3. $\{B_1, \ldots, B_n\} \subseteq b$ (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

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

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* [Robinson, 1965] coined the name "hyper tableaux".

It is a simple inductive consequence of the definition of hyper extension step that a branch is closed if and only if it ends in a negative literal. We prefer to have the definition slightly more general than needed in order not to have to change the hyper extension step rule below, where this situation changes in presence of the *atomic cut* inference rule. Also with view at the analytic cut we define slightly more general:

Definition 6 (Branch Semantics). As usual, we represent an interpretation $\mathcal{I}$ for given domain Σ as the set $\{A \in \Sigma \mid \mathcal{I}(A) = \mathit{true},\ A \text{ atom}\}$. *Minimality* of interpretations is defined via set-inclusion.

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

Definition 7 (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 $T_1, \ldots, T_n, \ldots$ 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 (cf. Def. 7), 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.

Example 8 (Hyper Tableau Derivation). The following is the clause set generated from our running Example 4.

OR1:	$ab(or1), high(or1, i1), high(or1, i2) \leftarrow high(or1, o)$
	$ab(or1), high(or1, o) \leftarrow high(or1, i1)$
	$ab(or1), high(or1, o) \leftarrow high(or1, i2)$
INV1:	$ab(inv1) \leftarrow high(inv1, o), high(inv1, i)$
	$ab(inv1), high(inv1, i), high(inv1, o) \leftarrow$
INV2:	$ab(inv2) \leftarrow high(inv2, o), high(inv2, i)$
	$ab(inv2), high(inv2, i), high(inv2, o) \leftarrow$
CONN1:	$high(or1, i1) \leftarrow high(inv1, o)$
	$high(inv1, o) \leftarrow high(or1, i1)$
CONN2:	$high(or1, i2) \leftarrow high(inv2, o)$
	$high(inv2, o) \leftarrow high(or1, i2)$
LOW_INV1_I:	$\leftarrow high(inv1, i)$
LOW_INV2_I:	$\leftarrow high(inv2, i)$
OBSERVATION:	$\leftarrow high(or1, o)$

Figure 3 contains a derivation. Each open branch corresponds to a partial model. 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 is it necessary to include $ab(or1)$ into the model. $high(or1, o)$ cannot be assumed, as this contradicts the observation $\leftarrow high(or1, o)$.

In order to make the generation of models efficient enough for the large benchmark circuits used in this paper, additional knowledge has to be used to guide this model generation. This is done by starting from a model of the correct behavior of the device *SD* and revising the model only where necessary. This idea of a "semantically guided " model generation has been introduced first in the DRUM-2 system [Fröhlich and Nejdl, 1996, Nejdl and Fröhlich, 1996].

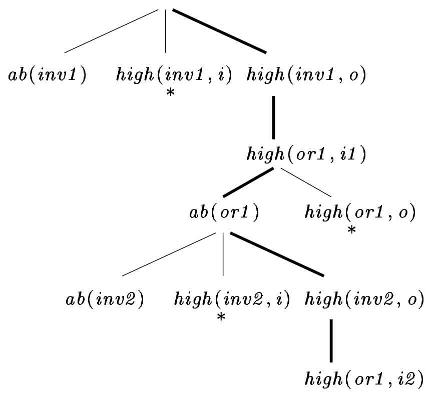

Fig. 3. Hyper derivation.

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 [Baumgartner *et al.*, 1996]. For our purposes of computing diagnosis (i.e. models), however, we need also a model completeness result:

Theorem 9 (Model Completeness of Hyper Tableaux.). *Let T be a hyper tableau such that every open branch is finished. Then, for every minimal model $\mathcal{I}$ of $\mathcal{S}$ there is an open branch b in T such that $\mathcal{I} \subseteq [\![b]\!]$.*

Proof. (Sketch) If no minimal model for $\mathcal{S}$ exists then the theorem holds vacuously. Otherwise let $\mathcal{I}$ be a minimal model for $\mathcal{S}$. It trivially holds that

$$\mathcal{S} \cup \mathcal{I} \cup \neg\overline{\mathcal{I}} \text{ is satisfiable,} \tag{1}$$

where $\overline{\mathcal{I}} := \Sigma \setminus \mathcal{I}$, and $\neg M := \{\neg A \mid A \in M\}$. It is not too hard to see that 1 is equivalent to $\mathcal{S} \cup \neg\overline{\mathcal{I}} \models \mathcal{I}$. (the minimality of $\mathcal{I}$ is essential here). This holds if and only if

$$\mathcal{S} \cup \neg\overline{\mathcal{I}} \cup \{ \bigvee_{A \in \mathcal{I}} \neg A\} \text{ is unsatisfiable.} \tag{2}$$

Hence, by completeness of Hyper tableaux there is a refutation of this clause set. Further, by 1, the subset $\mathcal{S} \cup \neg\overline{\mathcal{I}}$ is satisfiable. Hence, in any hyper tableau refutation the clause $\bigvee_{A \in \mathcal{I}} \neg A$ must be at used once for an extension step, say at branch b. But, by definition of hyper extension step this is possible only if the complementary literals are on the branch b, i.e. $\mathcal{I} \subseteq \mathrm{lit}(b)^+$. We can omit from the refutation all extension steps with $\bigvee_{A \in \mathcal{I}} \neg A$, as well as all extension steps with the negative unit clauses $\neg\overline{\mathcal{I}}$. The result is a hyper derivation from $\mathcal{S}$ alone. Now, either the branch b is finished, and the theorem is proven, or otherwise the derivation can be continued so that at least one open finished branch b'' with $\mathrm{lit}(b) \subseteq \mathrm{lit}(b'')$ comes up. Reason: otherwise every such extension b'' of b would be closed, meaning that we could find a refutation of $\mathcal{S} \cup \neg\overline{\mathcal{I}}$ alone, which by soundness of hyper tableau contradicts the satisfiability of $\mathcal{S} \cup \neg\overline{\mathcal{I}}$. Thus, b'' is the desired branch with $\mathcal{I} \subseteq [\![b'']\!]$.

4 The DRUM-2 System

Since Reiter's seminal paper [Reiter, 1987], several generic systems for model–based diagnosis have been developed using logical inference, assumption based truth maintenance and conflicts as their underlying principles (see [de Kleer and Williams, 1987] and many others). Efficiency problems due to administration overhead inherent to this approach have only recently been solved [Raiman *et al.*, 1993]. DRUM-2 ([Fröhlich and Nejdl, 1996, Nejdl and Fröhlich, 1996]) has emerged from a different line of research, where models serve as a data structure for the reasoning process. DRUM-2 has adapted and extended implementation ideas from model-based belief revision systems ([Chou and Winslett, 1994]).

The basic idea of DRUM-2 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 $SD \cup OBS$, but DRUM-2 uses I_0 to guide the search for models of $SD \cup OBS$. The models are computed iteratively by inverting truth values of literals in I_0 which contradict formulas in $SD \cup OBS$. We will now describe DRUM-2's algorithm intuitively using a simple circuit as an example. A formalization of DRUM-2 using belief revision vocabulary can be found in [Fröhlich and Nejdl, 1996].

We will now show how to compute diagnoses under the single fault assumption. n–Fault–Assumptions are a concept of the DRUM-2 diagnosis engine; they are therefore not represented as part of the theory. Figure 4 shows the steps performed by DRUM-2 during the search for consistent models of $SD \cup OBS$ from our example. In the first step the output observation $\neg high(or1, o)$ is incorporated into I_0 by deleting $high(or1, o)$ from the interpretation. The new interpretation I_1 contradicts the for-

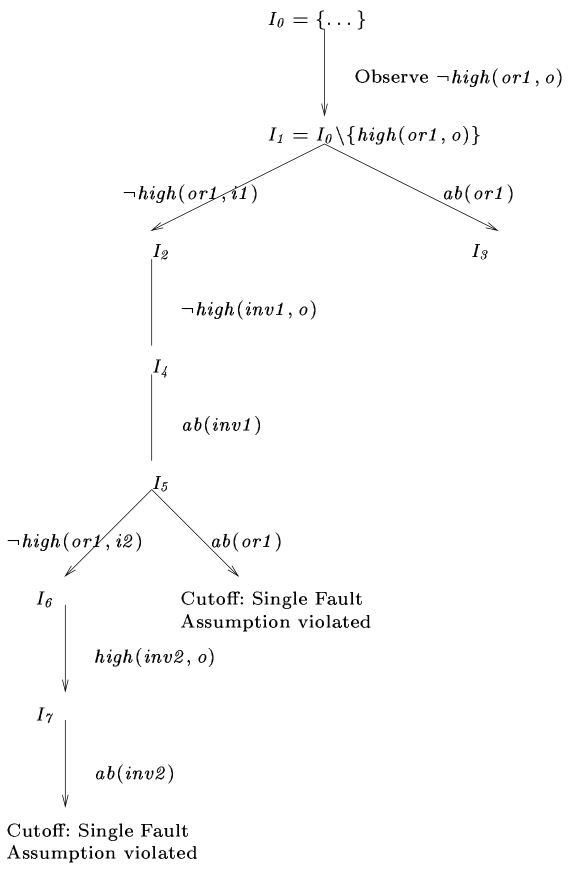

Fig. 4. "Repairing" an interpretation in DRUM-2.

mula *OR1*. There are two possible repair steps, which can remove the violation of this formula: removing $high(or1, i1)$ from the interpretation (leading to I_2) or adding $ab(or1)$ (leading to I_3). Since $I_3 \models SD \cup OBS$ we have found a diagnosis $I_3|Ab| = \{Ab(or1)\}$. In the left branch of the tree the search for diagnoses continues. However, since both inverters would have to be abnormal to explain the low voltage at the output of the inverter no other single fault diagnosis is found.

The changes to the model performed by DRUM-2 are focused by the initial interpretation I_0. Using this simple mechanism DRUM-2 is currently one of the fastest generic systems for model–based diagnosis as recently reported in [Nejdl and Fröhlich, 1996].

4.1 Effects of the Initial Interpretation

The small circuit in figure 2.2 is fine as a minimal example for clarifying the algorithms throughout this paper. However, it is too small to show the focusing effect of the initial interpretation I_0. In the slightly larger example depicted in figure 5 the computation of DRUM-2 would be exactly the same as in the smaller circuit. The reason for this efficiency gain is that because of the initial interpretation only those gates appear in the revision tree, which influence the abnormal observation. Note, that the benefit gained by using an initial interpretation is more than saving the computation of the expected output values. The computation of the initial model is just one deterministic simulation of the circuit under the assumption that all components are working. In figure 5 it is obvious that an uninformed procedure would have to follow several useless alternatives during the search for models, i.e. assume that *inv0*, *or0*, *inv2*, or *or2* are faulty. In fact, it has been shown in [Nejdl and Giefer, 1994] that the use of an initial model leads to a constant diagnosis time for a circuit consisting of n sequentially connected full adders, whereas the diagnosis time of uninformed algorithms is quadratic in n.

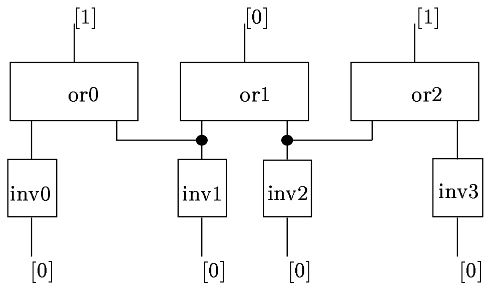

Fig. 5. A slightly larger circuit

5 Formalizing the Diagnosis Task with Semantic Hyper Tableaux

Recall from Section 4 that DRUM-2 uses an initial interpretation I_0 to focus on certain clauses from the clause set to be candidates for extending or generating new models. In this section we discuss how to incorporate initial interpretations into the hyper tableaux 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.

5.1 Initial Interpretations via Cuts

The use of an initial interpretation in DRUM-2 can be approximated in the hyper tableau calculus by the introduction of an additional inference rule, the *atomic cut rule*.

Definition 10. 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. 5).

Note that in regular derivations 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 derivation:

Definition 11. An *initial tableau* for an interpretation I_0 is given by a tableau which is constructed by a regular derivation where only atomic cuts with atoms from I_0 are applied, as long as possible.

The branches of an initial tableau for an interpretation I_0 consist obviously of all interpretations with atoms from I_0. In our running example the initial interpretation is

$$I_o = \{high(inv1, o), high(inv2, o), high(or1, i1), high(or1, i2), high(or1, o)\} \ .$$

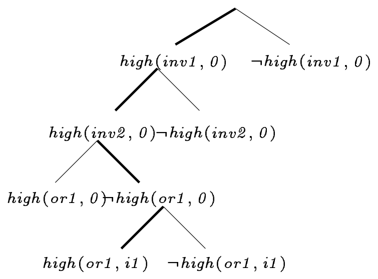

Fig. 6. Initial tableau.

A part of the initial tableau for this I_0 is depicted in Figure 6. Note that the highlighted branch corresponds to the highlighted part in Figure 3. If this branch is extended in successive derivation steps the diagnosis $ab(or1)$, which was contained in the model from Figure 3 can be derived as well.

Note that the cuts introduce negative literals into a branch. The Definition 6 of branch semantics applies to the calculus with cut as well: the interpretation associated to 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 derivation. Since this initial tableau represents semantics, we call tableaux from such a derivation *semantic hyper tableaux*.

Definition 12 (Semantic Hyper Tableau – SHT). A *semantic hyper tableau* for I_0 and $\mathcal{S}$ is a hyper tableau which is generated according to Definition 5, except that the empty tableau in the initialization step is replaced by an initial tableau for I_0. The definition of derivation (Def. 7) and its properties are adapted accordingly.

It is easy to derive an open tableau starting from the initial tableau for I_0 in Figure 6, such that it contains the model from Figure 3.

Refutational completeness of semantic hyper tableaux is not an immediate consequence of the completeness result given in [Baumgartner *et al.*, 1996]; the regularity condition does not allow to put any open tableau generated by a hyper derivation underneath an initial tableau, since this might lead to more than one occurrence of an atom introduced by a cut. But it still holds:

Theorem 13 (Refutational Completeness of SHT). *Every open finished branch in a semantic hyper tableau for I_0 and $\mathcal{S}$ is a model for $\mathcal{S}$. This holds in particular for the selected open branch $f(T_n)$ of the last tableau in any semantic hyper tableau derivation from I_0 and $\mathcal{S}$.*

The model completeness however follows directly from the corresponding Theorem 9:

Proposition 14 (Model Completeness of SHT). *Let T be a semantic hyper tableau for I_0 and $\mathcal{S}$, such that every open branch is finished. Then, for every minimal model $\mathcal{I}$ of $\mathcal{S}$ there is an open branch b in T such that $\mathcal{I} \subseteq [\![b]\!]$.*

5.2 Initial Interpretations via Renaming

The just defined "semantical" account for initial interpretations via cut is unsuited for realistic examples. This is, because all the $2^n - 1$ possible deviations from the initial interpretation will have to be investigated as well. Hence, 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 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 step be 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 the just described 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 15 (I-transformation). Let $C = L_1 \vee \cdots \vee L_n$ 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 $\mathcal{S}$* 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 $A \in I$.

As explained informally above, the branch semantics of tableaux derived from a renamed, i.e. I_0-transformed clause set, is changed to assign *true* to every atom from I_0, unless its negation is on the branch. This is a formal definition:

$$[\![b]\!]^I = (I \setminus \{A \mid neg\_A \in \mathrm{lit}(b)\}) \cup (\mathrm{lit}(b) \setminus \{neg\_A \mid neg\_A \in \mathrm{lit}(b)\})$$

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

Theorem 16. *Let T be a semantic hyper tableau for $\mathcal{S}$ and I_0 where every open branch is finished; let T^{I_0} be a hyper tableau for the I_0-transformation of $\mathcal{S}$ where every open branch is finished. Then, for every open branch b^{I_0} in T^{I_0} there is an open branch b in T such $[\![b^{I_0}]\!]^{I_0} = [\![b]\!]$. 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 sole 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 Implementation and Experiments

We have implemented a proof procedure for the hyper tableaux calculus of [Baumgartner *et al.*, 1996], 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 [Manthey and Bry, 1988]. Initially let T be a tableau consisting of the root node only. 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. 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 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 preferably select clauses 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, is superior for the benchmark circuits.

For further improvements of the proof procedure, such as *factorization* and *level cut* see [Baumgartner *et al.*, 1996].

Adaption for the Diagnosis Task. Recall that our diagnosis task requires to bias the proof search with hyper tableau in two ways: incorporation of the initial interpretation, and implementing the n-faults assumption (cf. Section 2.2). While the former is treated by renaming predicates in the input clause set (Section 5), 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

$$\leftarrow ab(C_1), \ldots, ab(C_{n+1}) \qquad \text{for } C_i \in COMP,\ C_i \neq C_j, \\ \text{where } 1 \leq i,j \leq n+1 \text{ and } i \neq j.$$

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.

No more changes to the basic proof procedure are required.

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. We are confident that this could be improved considerably by using standard techniques for propositional logic provers (Gallier/Downing algorithm). Some operations would have constant instead of linear complexity then. In fact, this pay-off is demonstrated in the DRUM-2 system.

Despite of these deficiencies, NIHIL is quite efficient due to a notable improvement over the more naive previous Prolog implementation described in [Baumgartner *et al.*, 1996]. It concerns the test whether a candidate clause $H \leftarrow B$ is applicable to the selected branch b, i.e. whether $B \subseteq b$ holds. The naive Prolog implementation walks through the input clauses and applies the test to each candidate until an applicable one is found. The test itself is of complexity $|B| \cdot |b|$. NIHIL improves on this by driving the search not be the clauses, but by the branch: as soon as a new leaf, say A, is added to a branch, indexing is used to determine those clauses having A in the body. In each of those clauses, this information is stored. Using this scheme, the applicability test for a given clause becomes a constant complexity operation then[2].

Of course, this idea is not new. For the propositional case it is due to Gallier/Downing; the RETE algorithm can be seen as a generalization towards first-order logic. However, expressed in our terminology, RETE has the restriction that it could perform only *matching* from the body literals towards the branch literals. Since we further generalized the basic idea of RETE towards true *unification*, and further we avoid severe deficiencies in the RETE algorithm, we consider our technique as an original contribution.

Experiments. For our experiments we ran parts of the ISCAS-85 benchmarks [Isc, 1985] from the diagnosis literature. This benchmark suite includes combinatorial circuits from 160 to 3512 components. Table 7 describes the characteristics of the circuits we tested. 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 7 on the right. *Time* denotes proof time proper in seconds, and thus excludes time for reading in and setup (which

Name	*# Gates*	*# Inputs*	*# Outputs*	*Time*	*# Clauses*	*# Steps*
C499	202	41	32	2	1685	2050
C880	383	60	26	1	2776	158
C1355	546	41	32	40	3839	21669
C2670	1193	233	140	4	8260	425
C3540	1669	50	22	1403	10658	253543
C5315	2307	178	123	13	16122	3024

Fig. 7. ISCAS'85 Circuits and NIHIL results.

[2] For instance, the Pigeonhole examples are proven much faster: while the Prolog implementation needs 260 seconds for Pigeon-6-in-5, NIHIL needs only 0.7 seconds. Of course, the same tableau is constructed.

is less than about
10 seconds in any case). The times are taken on a SparcStation 20. # *Clauses* is the number of input clauses; # *Steps* denotes the number of hyper extension steps to obtain the final tableau. We emphasize that the results refer to the clause sets with renamed predicates according to Section 5. Without renaming, and thus taking advantage of the initial interpretation, only c499 was solvable (in 174 seconds); all other examples could not be solved within 2 hours, whatever flag settings/heuristic we tried!

7 Conclusions

In this paper 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 the main optimization technique in the diagnostic reasoning engine DRUM-2 and showed how to apply this technique within a hyper tableaux calculus. The resulting theorem prover NIHIL very efficiently diagnoses 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 intersting to characterize the computed models more exactly. Further work will also include a closer examination of multiple faults (n-fault assumption) and further efficiency improvements.

References

[Baumgartner *et al.*, 1996] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *JELIA 96*. European Workshop on Logic in AI, Springer, LNCS, 1996. (Long version in: *Fachberichte Informatik*, 8–96, Universität Koblenz-Landau).

[Chou and Winslett, 1994] T. S-C. Chou and M. Winslett. A model–based belief revision system. *Journal of Automated Reasoning*, 12:157–208, 1994.

[Console and Torasso, 1991] Luca Console and Pietro Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3):133–141, 1991.

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

[Fröhlich and Nejdl, 1996] Peter Fröhlich and Wolfgang Nejdl. A model–based reasoning approach to circumscription. In *Proceedings of the 12th European Conference on Artificial Intelligence*, 1996.

[Isc, 1985] The ISCAS-85 Benchmarks. http://www.cbl.ncsu.edu/www/CBL_Docs/iscas85.html, 1985.

[Manthey and Bry, 1988] R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In *Proc. 9th CADE*. Argonnee, Illinois, Springer LNCS, 1988.

[Nejdl and Fröhlich, 1996] 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.

[Nejdl and Giefer, 1994] Wolfgang Nejdl and Brigitte Giefer. DRUM:Reasoning without conflicts and justifications. In *5th International Workshop on Principles of Diagnosis (DX-94)*, pages 226–233, October 1994.

[Raiman *et al.*, 1993] 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.

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

[Robinson, 1965] J. A. Robinson. Automated deduction with hyper-resolution. *Internat. J. Comput. Math.*, 1:227–234, 1965.