Research Report DBAI-TR-98-20

The Diagnosis Frontend of the dlv System

Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer

`eiter@kr.tuwien.ac.at`, {`faber`,`leone`,`pfeifer`}`@dbai.tuwien.ac.at`

Institut für Informationssysteme,
Technische Universität Wien,
A-1040 Vienna, Austria

Abstract

This paper presents the Diagnosis Front-End of `dlv`, which is a knowledge representation system under development at the *Technische Universität Wien*. The kernel language of the system is an extension of disjunctive logic programming (DLP) by integrity constraints; it offers front-ends to several advanced knowledge representation formalisms. The formal model of diagnosis employed in the front-end includes both abductive diagnosis (over DLP theories) and consistency-based diagnosis. For each of the two diagnosis modalities, generic diagnoses, single error diagnoses, and subset minimal diagnoses are considered. We illustrate the use of the front-end by showing the `dlv` encodings of several diagnosis problems. Thereafter, we discuss implementation issues. Diagnostic reasoning is implemented on the `dlv` engine through suitable translations of diagnostic problems into disjunctive logic programs, such that their stable models correspond to diagnoses. For the six kinds of diagnostic reasoning problems emerging from above, such reductions are provided.

Keywords: *Disjunctive Logic Programming, Diagnosis, Abduction, Stable Model*

1 Introduction

During the last years, many efforts have been spent on the development of formal theories for advanced knowledge representation, and implementations of formal systems tailored to solving particular KR problems have been achieved. However, the development of efficient systems which are applicable to a wider range of KR problems is still at an early stage.

A step in this direction is the dlv[1] system, which is actively being developed at the Institut für Informationssysteme, Technische Universität Wien. This system is based on the function-free (datalog) fragment of disjunctive logic programming [28, 12], and embodies a general purpose disjunctive datalog engine. The implementation relies on algorithms and results presented in [25], which have been subsequently refined and improved. Due to the expressive power of the language [12, 2], the system is capable of efficiently expressing problems whose complexity is beyond the scope of non-disjunctive logic programming. This is exploited in domain specific frontends for various application domains. A comparison of dlv to similar systems is given in [16].

In this paper, we present the Diagnosis Front-End of dlv. While there are many specialized efficient systems for logic-based diagnosis available (see this volume), we stress at this point that our goal is not the development of a high-speed diagnostic system which competes with the fastest systems in performance. Rather, our intent is to demonstrate that different formalizations of logic-based diagnosis can be compiled at a conceptual level into the dlv engine, and that moderate-sized problems can be solved this way in practice. This qualifies dlv as a ready-to-use tool for solving ad hoc such problems, and as a testbed for experimenting with different forms of diagnosis. (See Section 7 for download information.)

We proceed in this paper as follows. We first give in Section 2 a brief review of the concepts of disjunctive logic programming which are used for the kernel language of dlv. After that, we describe in Sections 3 and 4 the different formal models of diagnosis that are realized in the front-end. It supports both abductive diagnosis [31, 5], adapted to the semantics of logic programming [22, 11], and consistency-based diagnosis [34, 9].

In our abductive model of diagnosis (Section 3), the theory is a disjunctive logic program, whose semantics is given by the set of its stable models [20]. This form of diagnostic reasoning has been recently proved to be highly expressive, as it is able to represent even problems located at the third level of the polynomial hierarchy [11]. On the other hand, the consistency-based diagnosis model (Section 4) refers to theories of first-order logic under classical semantics, similarly to the original definition by Reiter [34]. For each of the two diagnosis modalities (abduction and consistency-based), generic diagnoses, single-error diagnoses and subset minimal diagnoses are

[1] dlv stands for Datalog$^\vee$, i.e. Datalog with disjunction.

considered in our model. We then illustrate the various kinds of diagnostic reasoning supported in `dlv` by a number of examples, which demonstrate the power and the simplicity of our model of diagnosis.

In Section 5, we present six translation algorithms (one for each kind of diagnostic reasoning problem emerging from the discussion above) which rewrite diagnosis problems into disjunctive logic programs, such that each stable model of such a program corresponds to precisely one diagnosis, and each diagnosis has some corresponding stable model. In particular, for every stable model a unique corresponding diagnosis exists. On the other hand, for every diagnosis a corresponding stable model of the program exists. This part can be viewed as the central contribution of this paper. Running such a program on `dlv`, we obtain all diagnoses of the input diagnosis problem.

We show then in Section 6 the overall architecture of the system, and conclude the paper in Section 7.

2 Preliminaries on Disjunctive Logic Programming

For a full background on (disjunctive) logic programming, we refer to [27], [3], [28] and [12]. For an overview of semantics for (disjunctive) logic programs, see [10]. The language described in this paper (Datalog$^{\vee,\neg,s}$) has been described in [2].

A *term* is either a constant or a variable[2]. An *atom* is $a(t_1, ..., t_n)$, where a is a *predicate* of arity n and $t_1, ..., t_n$ are terms. A *literal* is either a *positive literal* p or a *negative literal* $\neg p$, where p is an atom.

A *(disjunctive datalog) rule* r is a clause of the form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k, \neg b_{k+1}, \cdots, \neg b_{k+m}$$

where $n \geq 1,\ k, m \geq 0$, and $a_1, \ldots, a_n, b_1, \ldots, b_{k+m}$ are function-free atoms. The disjunction $a_1 \vee \cdots \vee a_n$ is the *head* of r, while the conjunction b_1, ..., b_k, $\neg b_{k+1}$, ..., $\neg b_{k+m}$ is the *body* of r.

If $n = 1$, then r is *normal* (i.e., $\vee$-free); if $m = 0$, then r is *positive* (or $\neg$-free). A *disjunctive datalog program* (also called Datalog$^{\vee,\neg}$ *program*) $\mathcal{P}$ is a finite set of disjunctive datalog rules; $\mathcal{P}$ is *normal* (Datalog$^{\neg}$ *program*) if all its rules are normal; $\mathcal{P}$ is *positive* (Datalog$^{\vee}$ *program*) if all its rules are positive.

We denote by $H(r)$ (resp. $B^+(r)$, $B^-(r)$) the set of head atoms (resp. positive body literals, negative body literals) of r. In general, for any set X of literals, X^+ denotes the set of positive and X^- denotes the set of negative literals in X; further, we denote by $\neg.X$ the set $\{\neg\, a \mid a \in X^+\} \cup \{a \mid \neg\, a \in X^-\}$.

An *(integrity) constraint* is an expression of the form $\leftarrow\ b_1, \ldots, b_k$, where b_i, $1 \leq\ i \leq k$, is a literal (i.e., it is a rule with empty head).

[2]Note that function symbols are not considered in this paper.

In general, a syntactic object is *ground* if it does not contain variables.

As usual in the context of deductive databases, we assume that rules are *safe*, i.e., each variable occurring in a rule also occurs in a positive body literal of that rule.

We define a *Datalog*$^{\vee,\neg,s}$ *program* (often simply program) $\mathcal{P} = \mathcal{P}' \cup S$, where $\mathcal{P}'$ is a Datalog$^{\vee,\neg}$ program and S a (possibly empty) set of integrity constraints.

We denote by $U_\mathcal{P}$ the Herbrand universe (the set of all constants in $\mathcal{P}$), by $B_\mathcal{P}$ the Herbrand base (all instantiations of predicates in $\mathcal{P}$ with terms of $U_\mathcal{P}$), and by $ground(\mathcal{P})$ the instantiation of all rules and constraints in $\mathcal{P}$ with terms from $U_\mathcal{P}$. An (Herbrand) interpretation I for $\mathcal{P}$ is a subset of $B_\mathcal{P}$ (representing the atoms which are *true* in I). A ground positive literal A is true w.r.t. I iff $A \in I$; otherwise it is false. A ground negative literal $\neg A$ is true (false) w.r.t. I iff A is false (resp. true) w.r.t. I.

A ground rule r is satisfied by an interpretation I if the head is true or if the body is false. A constraint is treated as a rule with empty head (it is satisfied iff its body is false).

An interpretation M is a model of a Datalog$^{\vee,\neg}$ program $\mathcal{P}$, if all rules in $ground(\mathcal{P})$ are satisfied by M.

Minker proposed in [29] a model-theoretic semantics for (positive) Datalog$^{\vee}$ programs $\mathcal{P}$, which assigns $\mathcal{P}$ the set $\mathrm{MM}(\mathcal{P})$ of its *minimal models*, where a model M for $\mathcal{P}$ is minimal, if no proper subset of M is a model for $\mathcal{P}$. Accordingly, the program $\mathcal{P} = \{a \vee b \leftarrow\}$ has the two minimal models $\{a\}$ and $\{b\}$, i.e., $\mathrm{MM}(\mathcal{P}) = \{\ \{a\},\ \{b\}\ \}$.

The more general stable model semantics also applies to programs with negation. Given a Datalog$^{\vee,\neg}$ program $\mathcal{P}$ and a total interpretation I, the *Gelfond-Lifschitz transform* [19, 33] of $\mathcal{P}$ w.r.t. I, is the positive program

$$\mathcal{P}^I = \{H(r) \leftarrow B^+(r) \mid \begin{array}{l} r \in ground(\mathcal{P}), \\ \neg.B^-(r) \cap I = \emptyset\}. \end{array}$$

Then, an interpretation I is a *stable model* for $\mathcal{P}$, if $I \in \mathrm{MM}(\mathcal{P}^I)$ [33, 20]; the set of all stable models for $\mathcal{P}$ is denoted by $STM(\mathcal{P})$.

For extending the definition to full Datalog$^{\vee,\neg,s}$ programs we only have to consider constraints [2]:

Definition 1 *A* stable model *for a program $\mathcal{P} = \mathcal{P}' \cup S$ ($\mathcal{P}'$ a Datalog$^{\vee,\neg}$ program, S a set of integrity constraints) is a* stable model for $\mathcal{P}'$ *which satisfies all constraints in $ground(S)$.* □

Finally, we need an operator for reasoning over Datalog$^{\vee,\neg,s}$ programs.

Definition 2 *We define the consequence operator "$\models$" as the brave (or credulous) reasoning operator under the stable model semantics: For a Datalog$^{\vee,\neg,s}$ program $\mathcal{P}$ and a set of literals X, $\mathcal{P} \models X$ holds, if there exists a stable model M of $\mathcal{P}$, such* *that* $X^+ \subseteq M$ *and* $\neg.X^- \cap M = \emptyset$ *hold. In other words: There is at least one stable model* M *of* $\mathcal{P}$*, in which all literals of* X *are true.* $\square$

Example 1 *Consider the Datalog*$^{\vee,\neg,s}$ *program* $\mathcal{P}_e = \mathcal{P}'_e \cup S_e$*:*

$$\mathcal{P}'_e = \{\mathtt{a} \leftarrow\ , \mathtt{b} \vee \mathtt{c} \leftarrow \mathtt{a}\ \}$$

$$S_e = \{\leftarrow \mathtt{c}\ \}$$

$\mathcal{P}'_e$ *has two stable models:* $STM(\mathcal{P}'_e) = \{\{\mathtt{a}, \mathtt{b}\}, \{\mathtt{a}, \mathtt{c}\}\}$
$\{\mathtt{a}, \mathtt{c}\}$ *does not satisfy the constraint in* S_e*, so the only stable model of* $\mathcal{P}_e$ *is* $\{\mathtt{a}, \mathtt{b}\}$*:*

$$STM(\mathcal{P}_e) = \{\{\mathtt{a}, \mathtt{b}\}\}$$

Therefore the following consequences are valid:

$$\begin{aligned}
\mathcal{P}_e &\models \emptyset \\
\mathcal{P}_e &\models \{\mathtt{a}\} \\
\mathcal{P}_e &\models \{\mathtt{b}\} \\
\mathcal{P}_e &\models \{\mathtt{a}, \mathtt{b}\}
\end{aligned}$$

$\triangle$

3 Abductive Diagnosis over DLP Theories

Abductive diagnostic reasoning has been widely studied in the literature in the last years (see, e.g., [31, 30, 23, 5, 22, 11]).

Intuitively, a diagnostic problem consists of a description of the system to be diagnosed, observations of the actual state of the system, and possible reasons for effects, which are usually errors.

Definition 3 *We define an* abductive diagnostic problem (ADP) $\mathcal{P}$ *as a triple* $\langle H, T, O\rangle$*, where:*

- H *is a set of ground atoms and is referred to as* hypotheses.
- T *is a Datalog*$^{\vee,\neg,s}$ *program as described in Section 2 and is referred to as* theory*.*
- O *is a set of ground literals and is referred to as* observations.

$\square$

The solutions to ADPs are referred to as abductive diagnoses. In the following subsections, we describe several notions of reasoning over abductive diagnostic problems.

3.1 Generic Abductive Diagnosis

In the case of finding generic diagnoses, any subset of the hypotheses which "explains" the observations based on the system description is considered a diagnosis. By an explanation in the abductive case we understand that the diagnosis together with the system model gives reason for the observations.

Definition 4 *Given an ADP $\mathcal{P} = \langle H, T, O \rangle$, a* generic diagnosis *is a set $\Delta \subseteq H$, such that $T \cup \Delta \models O$ holds.* □

Note that our definition incorporates the brave reasoning consequence operator. In the literature some definitions are based on cautious reasoning, i.e., O must be true in *all* (stable) models of $T \cup \Delta$. See for example [11] for a variety of definitions.

Our first example shows how to use disjunction in the system description:

Example 2 *Consider an electric circuit, as shown in Fig. 1, consisting of a simple stove with two hot-plates wired in parallel and a control light, which is on if at least one of the plates is in operation. Each plate has a fuse, and it is known that one of them cannot stand much current and will melt if the current gets high, but it is not known which.*

We assume that there are several equally likely causes of malfunction: The power supply may fail (in this case `power_off` *holds), the bulb may be broken (*`broken_bulb` *holds), or the current may be (unusually) high (*`high_curr` *holds). So our hypotheses are:*

$$H_{st} = \{\texttt{broken\_bulb}, \texttt{power\_off}, \texttt{high\_curr}\}$$

Now for the system description T_{stove}:

$$\begin{array}{l}
\texttt{light_off} \leftarrow \texttt{power_off} \\
\texttt{light_off} \leftarrow \texttt{broken_bulb} \\
\texttt{light_off} \leftarrow \texttt{melted_fuse}_1, \texttt{melted_fuse}_2 \\
\texttt{melted_fuse}_1 \vee \texttt{melted_fuse}_2 \leftarrow \texttt{high_curr} \\
\texttt{hot_plate}_1 \leftarrow \neg\, \texttt{melted_fuse}_1, \neg\, \texttt{power_off} \\
\texttt{hot_plate}_2 \leftarrow \neg\, \texttt{melted_fuse}_2, \neg\, \texttt{power_off}
\end{array}$$

The first three rules describe situations under which the control light (emitted by the bulb) is off, namely, if the power fails, the bulb is broken, or both fuses are melted.

The fourth rule states that, on high current, a fuse will melt.

The last rules state that a hot-plate is heated if there is no power failure and the fuse is not melted. [3]

[3] To keep the example simple, we refrain from modeling switches for the hot-plates.

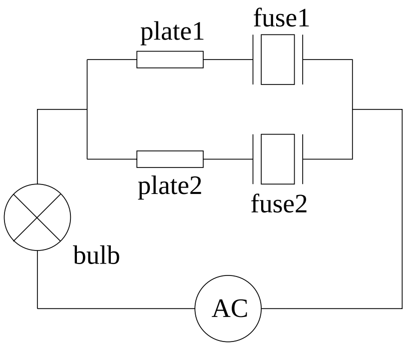

Figure 1: Simplified circuit diagram of a stove.

We observe that no light is emitted and that plate 1 is not hot:

$$O_{st} = \{\texttt{light\_off}, \neg\,\texttt{hot\_plate}_1\}$$

The goal is to find diagnoses to the ADP $\mathcal{P}_{st} = \langle H_{st}, T_{st}, O_{st}\rangle$*, which explain the current situation.*

Consider now $\Delta_1 = \{\texttt{power\_off}\}$*. Then* $\Delta_1 \cup T_{st}$ *has one stable model:*

$$M_1 = \{\texttt{power\_off}, \texttt{light\_off}\}$$

`light_off` *is in* M_1*, and* $\texttt{hot\_plate}_1$ *is not, so* Δ_1 *is a valid diagnosis.*

Let us examine whether $\Delta_2 = \{\texttt{broken\_bulb}\}$ *is a valid diagnosis.* $\Delta_2 \cup T_{st}$ *also has only one stable model:*

$$\begin{aligned} M_2 = \{&\texttt{broken\_bulb}, \texttt{light\_off}, \\ &\texttt{hot\_plate}_1, \texttt{hot\_plate}_2\} \end{aligned}$$

So Δ_2 *is not a valid diagnosis, since there is no model of* $\Delta_2 \cup T_{st}$ *in which* $\texttt{hot\_plate}_1$ *is not contained.*

On the other hand, if we augment Δ_2 *to*

$$\Delta_3 = \{\texttt{broken\_bulb}, \texttt{high\_curr}\}$$

the program $\Delta_3 \cup T_{st}$ *has two stable models:*

$$\begin{aligned} M_3 = \Delta_3 \cup \{&\texttt{light\_off}, \texttt{melted\_fuse}_1, \\ &\texttt{hot\_plate}_2\} \\ M_4 = \Delta_3 \cup \{&\texttt{light\_off}, \texttt{melted\_fuse}_2, \\ &\texttt{hot\_plate}_1\} \end{aligned}$$

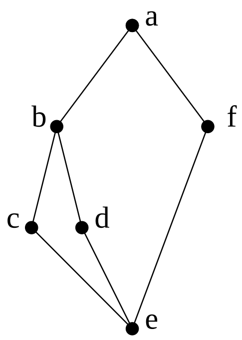

Figure 2: Topology of a point-to-point network.

One model (M_3) exists, in which `light_off` *is contained, but* $\texttt{hot\_plate}_1$ *is not, therefore Δ_3 is a valid diagnosis. Note that the observations do not hold in M_4, albeit it is sufficient that one model explains the observations since we consider brave reasoning.*

Other diagnoses for $\mathcal{P}_{st}$ are:

$$\{\texttt{power\_off}, \texttt{high\_curr}\}$$
$$\{\texttt{power\_off}, \texttt{broken\_bulb}\}$$
$$\{\texttt{power\_off}, \texttt{high\_curr}, \texttt{broken\_bulb}\}$$

△

The system description of the next example utilizes the notion of reachability which is easily defined in logic programming (through a recursive predicate) while it cannot be defined at all in classical first order logic.

Example 3 *Consider a computer network as depicted in Fig. 2 (the nodes denote computers and the arcs represent point-to-point connections).*

The network is described by facts of the form $\texttt{connected}(\texttt{x}, \texttt{y})$*, which state that there is a connection between node* `x` *and node* `y`*. The main part of the system description is the notion of* reachability*: A node is reachable from another node if there is a path between them and all computers on that path are online* ($\neg\,\texttt{off}(\texttt{X})$)*.*

$$\begin{array}{rl}
\texttt{reaches}(\texttt{X}, \texttt{X}) \leftarrow & \texttt{node}(\texttt{X}), \neg\,\texttt{off}(\texttt{X}) \\
\texttt{reaches}(\texttt{X}, \texttt{Z}) \leftarrow & \texttt{reaches}(\texttt{X}, \texttt{Y}), \\
& \texttt{connected}(\texttt{Y}, \texttt{Z}), \\
& \neg\,\texttt{off}(\texttt{Z})
\end{array}$$

The system description T_{net} consists of the facts of `connected` *and the above rules for* `reaches`*.*

The hypotheses are given by the assumptions that particular machines are offline:

$$H_{net} = \{\texttt{off(a)}, \texttt{off(b)}, \texttt{off(c)}, \texttt{off(d)}, \\ \texttt{off(e)}, \texttt{off(f)}\}$$

If we work on computer a *and fail to reach* e*, the observations are given as follows:*

$$O_{net} = \{\neg\, \texttt{off(a)}, \neg\, \texttt{reaches(a, e)}\}$$

Valid diagnoses are:

$$
\begin{array}{l}
\{\texttt{off(e)}\} \\
\{\texttt{off(e)}, \texttt{off(b)}\} \\
\{\texttt{off(e)}, \texttt{off(c)}\} \\
\{\texttt{off(e)}, \texttt{off(d)}\} \\
\{\texttt{off(e)}, \texttt{off(f)}\} \\
\{\texttt{off(e)}, \texttt{off(b)}, \texttt{off(c)}\} \\
\{\texttt{off(e)}, \texttt{off(b)}, \texttt{off(d)}\} \\
\{\texttt{off(e)}, \texttt{off(b)}, \texttt{off(f)}\} \\
\{\texttt{off(e)}, \texttt{off(b)}, \texttt{off(c)}, \texttt{off(d)}\} \\
\{\texttt{off(e)}, \texttt{off(b)}, \texttt{off(c)}, \texttt{off(f)}\} \\
\{\texttt{off(e)}, \texttt{off(b)}, \texttt{off(d)}, \texttt{off(f)}\} \\
\{\texttt{off(e)}, \texttt{off(b)}, \texttt{off(c)}, \texttt{off(d)}, \texttt{off(f)}\} \\
\{\texttt{off(e)}, \texttt{off(c)}, \texttt{off(d)}\} \\
\{\texttt{off(e)}, \texttt{off(c)}, \texttt{off(f)}\} \\
\{\texttt{off(e)}, \texttt{off(c)}, \texttt{off(d)}, \texttt{off(f)}\} \\
\{\texttt{off(e)}, \texttt{off(d)}, \texttt{off(f)}\} \\
\{\texttt{off(b)}, \texttt{off(f)}\} \\
\{\texttt{off(b)}, \texttt{off(f)}, \texttt{off(c)}\} \\
\{\texttt{off(b)}, \texttt{off(f)}, \texttt{off(d)}\} \\
\{\texttt{off(b)}, \texttt{off(f)}, \texttt{off(c)}\ \ \texttt{off(d)}\} \\
\{\texttt{off(c)}, \texttt{off(d)}, \texttt{off(f)}\}
\end{array}
$$

△

Example 3 shows that in general many diagnoses are in a sense redundant: If one error is enough to explain the observations, one may additionally assume several other hypotheses and still be able to explain the observations, but there is no information gained. So in general it is preferable to require some additional minimality criterion.

3.2 Single Error Abductive Diagnosis

A simple, but commonly used minimality criterion is to require that diagnoses are made up of exactly one hypothesis each.

Definition 5 *Given an ADP $\mathcal{P} = \langle H, T, O \rangle$, a generic diagnosis Δ of $\mathcal{P}$ is a* single error *diagnosis, if $|\Delta| = 1$ holds.* □

Example 4 *Reconsider Example 2:*
The only single error diagnosis for this example is $\Delta_1 = \{\texttt{power\_off}\}$.
Now reconsider Example 3:
The only single error diagnosis for this example is $\{\texttt{off(e)}\}$. △

At least in the case of Example 3 it is obvious that under single error assumption some potentially important diagnoses are missed (those which do not follow directly from a single error diagnosis by assuming additional errors).

Even worse, in many cases (for instance if the observations include $\neg\,\texttt{off(e)}$) a single error diagnosis does not exist.

3.3 Subset Minimal Abductive Diagnosis

A more sensible version of minimality criterion is to require that only those diagnoses should be considered, which do not contain any subset that is a diagnosis itself.

Definition 6 *Given an ADP $\mathcal{P} = \langle H, T, O \rangle$, a generic diagnosis Δ of $\mathcal{P}$ is a* subset minimal *diagnosis, if for every diagnosis Δ^* of $\mathcal{P}$ $\Delta^* \not\subset \Delta$ holds.* □

Example 5 *If we reconsider Example 2, we see that there are two subset minimal diagnoses: $\{\texttt{power\_off}\}$ and $\{\texttt{broken\_bulb}, \texttt{high\_curr}\}$.*
Now consider Example 3: There are three subset minimal diagnoses:

$$\begin{array}{l} \{\texttt{off(e)}\}, \\ \{\texttt{off(b)}, \texttt{off(f)}\} \\ \{\texttt{off(c)}, \texttt{off(d)}, \texttt{off(f)}\} \end{array}$$

△

4 Consistency-Based Diagnosis

Consistency-based diagnosis has been defined e.g. in [34, 9].

As in the abductive case, we first introduce the general problem formulation and in the following subsections we define several notions of consistency-based diagnosis.

Definition 7 *A* consistency-based diagnostic problem (CDP) *$\mathcal{P}$ is a triple $\langle H, T, O \rangle$, where:*

- *H is a set of ground atoms with predicate name ab (standing for* abnormal*) and is referred to as* hypotheses.

- *T is a set of first-order sentences and is referred to as* theory.
- *O is a set of ground literals and is referred to as* observations.

□

In this paper, only Herbrand interpretations (resp. models) and function-free theories where sentences are universally quantified are considered. Further, we assume that these sentences are written in clausal form so that they can be represented by Datalog$^{\vee,\neg,s}$ programs.

H is comprised of atoms of the form ab(c), meaning that the component c behaves abnormally.

4.1 Generic Consistency-Based Diagnosis

Definition 8 *Given an CDP $\mathcal{P} = \langle H, T, O \rangle$, a* generic diagnosis *is a set $\Delta \subseteq H$, such that $T \cup O \cup \Delta \cup \{\neg h | h \in H - \Delta\}$ is consistent in the classical sense.*

□

Note that the consistency-based approach requires a weaker property on the diagnoses compared to the abductive approach. Indeed, Definition 8 requires that $T \cup O \cup \Delta \cup \{\neg h | h \in H - \Delta\}$ is consistent in the classical sense (i.e., it has a model), while Definition 4 requires the existence of a *stable* model of $T \cup \Delta$ which entails O. Thus, every diagnosis of an ADP $\mathcal{P} = \langle H, T, O \rangle$ is also a diagnosis of the corresponding consistency-based diagnosis problem (provided that hypotheses do not occur in the heads of the rules), while the vice versa does not hold in general.

For comparisons between the abductive and consistency-based approaches, see [32, 24, 6].

Example 6 *Consider the circuit description of a full adder (depicted in Fig. 3)[4]:*

For the system description T_{add}, we first give definitions for the gates involved.

The and *gates:*

$$\begin{array}{l}
\mathtt{out(X,1)} \leftarrow \mathtt{and(X), in1(X,1), in2(X,1)}, \neg\ \mathtt{ab(X)} \\
\mathtt{out(X,0)} \leftarrow \mathtt{and(X), in1(X,0)}, \neg\ \mathtt{ab(X)} \\
\mathtt{out(X,0)} \leftarrow \mathtt{and(X), in2(X,0)}, \neg\ \mathtt{ab(X)}
\end{array}$$

The or *gates:*

$$\begin{array}{l}
\mathtt{out(X,0)} \leftarrow \mathtt{or(X), in1(X,0), in2(X,0)}, \neg\ \mathtt{ab(X)} \\
\mathtt{out(X,1)} \leftarrow \mathtt{or(X), in1(X,1)}, \neg\ \mathtt{ab(X)} \\
\mathtt{out(X,1)} \leftarrow \mathtt{or(X), in2(X,1)}, \neg\ \mathtt{ab(X)}
\end{array}$$

[4] This example is taken from [34].

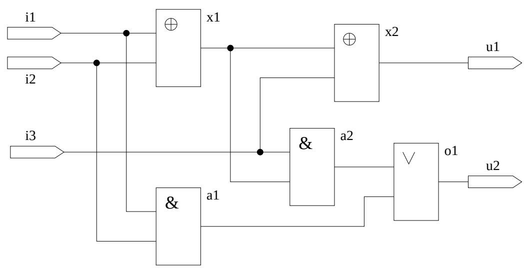

Figure 3: Circuit diagram of a full adder

The xor *gates:*

$$\begin{array}{l}
\mathtt{out}(\mathtt{X},1) \leftarrow \mathtt{xor}(\mathtt{X}), \mathtt{in1}(\mathtt{X},1), \mathtt{in2}(\mathtt{X},0), \neg\, \mathtt{ab}(\mathtt{X}) \\
\mathtt{out}(\mathtt{X},1) \leftarrow \mathtt{xor}(\mathtt{X}), \mathtt{in1}(\mathtt{X},0), \mathtt{in2}(\mathtt{X},1), \neg\, \mathtt{ab}(\mathtt{X}) \\
\mathtt{out}(\mathtt{X},0) \leftarrow \mathtt{xor}(\mathtt{X}), \mathtt{in1}(\mathtt{X},\mathtt{Y}), \mathtt{in2}(\mathtt{X},\mathtt{Y}), \neg\, \mathtt{ab}(\mathtt{X})
\end{array}$$

Then we specify that all of the in- and outputs are binary-valued:

$$\begin{array}{l}
\mathtt{gate}(\mathtt{X}) \leftarrow \mathtt{xor}(\mathtt{X}) \\
\mathtt{gate}(\mathtt{X}) \leftarrow \mathtt{and}(\mathtt{X}) \\
\mathtt{gate}(\mathtt{X}) \leftarrow \mathtt{or}(\mathtt{X}) \\
\\
\mathtt{in1}(\mathtt{X},0) \vee \mathtt{in1}(\mathtt{X},1) \leftarrow \mathtt{gate}(\mathtt{X}) \\
\mathtt{in2}(\mathtt{X},0) \vee \mathtt{in2}(\mathtt{X},1) \leftarrow \mathtt{gate}(\mathtt{X}) \\
\mathtt{out}(\mathtt{X},0) \vee \mathtt{out}(\mathtt{X},1) \leftarrow \mathtt{gate}(\mathtt{X})
\end{array}$$

Also, we ensure that an input or output must have only one associated truth value:

$$\begin{array}{l}
\leftarrow \mathtt{out}(\mathtt{X},0), \mathtt{out}(\mathtt{X},1) \\
\leftarrow \mathtt{in1}(\mathtt{X},0), \mathtt{in1}(\mathtt{X},1) \\
\leftarrow \mathtt{in2}(\mathtt{X},0), \mathtt{in2}(\mathtt{X},1)
\end{array}$$

After these general definitions, we model the circuit wiring:

$$\begin{aligned}
\mathtt{in1(a1,S)} &\leftarrow \mathtt{in1(x1,S)}\\
\mathtt{in1(x1,S)} &\leftarrow \mathtt{in1(a1,S)}\\[1em]
\mathtt{in2(a1,S)} &\leftarrow \mathtt{in2(x1,S)}\\
\mathtt{in2(x1,S)} &\leftarrow \mathtt{in2(a1,S)}\\[1em]
\mathtt{in2(x2,S)} &\leftarrow \mathtt{in1(a2,S)}\\
\mathtt{in1(a2,S)} &\leftarrow \mathtt{in2(x2,S)}\\[1em]
\mathtt{out(x1,S)} &\leftarrow \mathtt{in1(x2,S)}\\
\mathtt{in1(x2,S)} &\leftarrow \mathtt{out(x1,S)}\\[1em]
\mathtt{out(x1,S)} &\leftarrow \mathtt{in2(a2,S)}\\
\mathtt{in2(a2,S)} &\leftarrow \mathtt{out(x1,S)}\\[1em]
\mathtt{out(a1,S)} &\leftarrow \mathtt{in2(o1,S)}\\
\mathtt{in2(o1,S)} &\leftarrow \mathtt{out(a1,S)}\\[1em]
\mathtt{out(a2,S)} &\leftarrow \mathtt{in1(o1,S)}\\
\mathtt{in1(o1,S)} &\leftarrow \mathtt{out(a2,S)}
\end{aligned}$$

The hypotheses H_{add} are that each of the gates is abnormal (`ab`*):*

$$H_{add} = \{\mathtt{ab(x1)}, \mathtt{ab(x2)}, \mathtt{ab(a1)}, \mathtt{ab(a2)}, \mathtt{ab(o1)}\}$$

It is observed that for the input $\mathtt{i1}:1, \mathtt{i2}:0, \mathtt{i3}:1$ *the output is* $\mathtt{u1}:1, \mathtt{u2}:0$.[5]

$$\begin{aligned}
O_{add} = \{\ & \mathtt{in1(x1,1)}, \mathtt{in2(x1,0)}, \mathtt{in1(a2,1)},\\
& \mathtt{out(x2,1)}, \mathtt{out(o1,0)}\}
\end{aligned}$$

The smallest generic consistency-based diagnoses of $\mathcal{P}_{add} = \langle H_{add}, T_{add}, O_{add}\rangle$ are

$$\begin{aligned}
\Delta_1 &= \{\mathtt{ab(x1)}\}\\
\Delta_2 &= \{\mathtt{ab(x2)}, \mathtt{ab(o1)}\}\\
\Delta_3 &= \{\mathtt{ab(x2)}, \mathtt{ab(a2)}\}
\end{aligned}$$

These are in fact the subset minimal diagnoses; for generic diagnoses, an addition of any number of hypotheses to the diagnoses above still yields a diagnosis.

[5] To save space, we did not model the input and output elements seperately. Instead, we use the uppermost connections to each input and output in Fig. 3.

Note that an abnormal component does not need to show exactly dual behavior to its functioning counterpart, so assuming one additional component to be faulty does not necessarily mean that its output is reversed. If this was true, there would be less additional diagnoses.

The full set of all generic diagnoses can be described as:[6]

$$\{\Delta \cup X \mid \Delta \in \{\Delta_1, \Delta_2, \Delta_3\}, X \in P(H)\}$$

△

Again it is clear that many diagnoses are irrelevant. Indeed, the original definition in [34] includes a (subset) minimality criterion.

4.2 Single Error Consistency-Based Diagnosis

As in the abductive case, it is often feasible to look for single error diagnoses, i.e. diagnoses which label exactly one component as abnormal.

Definition 9 *Given a CDP $\mathcal{P} = \langle H, T, O\rangle$, a generic diagnosis Δ of $\mathcal{P}$ is a* single error *diagnosis, if $|\Delta| = 1$ holds.* □

Example 7 *Reconsider Example 6: The only single error diagnosis for $\mathcal{P}_{add}$ is* $\{\texttt{ab(x1)}\}$*.* △

4.3 Subset Minimal Consistency-Based Diagnosis

In the field of consistency-based diagnostic reasoning, most definitions of diagnoses include a subset minimality criterion.

Definition 10 *Given a CDP $\mathcal{P} = \langle H, T, O\rangle$, a generic diagnosis Δ of $\mathcal{P}$ is a* subset minimal *diagnosis, if for every diagnosis Δ^* of $\mathcal{P}$ $\Delta^* \not\subset \Delta$ holds.* □

Example 8 *As already stated in Example 6, the subset minimal diagnoses for $\mathcal{P}_{add}$ are Δ_1, Δ_2 and Δ_3.*

△

[6] $P(X)$ denotes the powerset of X.

5 From Diagnoses to DLP Problems

In this section, we provide several algorithms which transform diagnostic problems (as defined in Section 3) into Datalog$^{\vee,\neg,s}$ programs (as defined in Section 2).

The transformation algorithms for consistency-based diagnostic problems are presented in Appendix A.

We remark that, in some cases, we impose syntactic restrictions on the diagnosis theory in order to achieve better performance in the implementation resp. due to complexity issues (i.e., cases which are harder than Σ_2^P and thus cannot be expressed by Datalog$^{\vee,\neg,s}$).

A different approach to the transformation of abductive logic programs into disjunctive logic programs can be found in [21]. Sakama and Inoue describe in [35] a transformation from abductive logic programs to disjunctive logic programs under the possible model semantics, which exploits a similar idea as our transformation in the generic case. To our knowledge, other translations mapping subset minimal diagnoses to stable models of disjunctive logic programs have not been discussed elsewhere.

The basic abductive translation is performed by the function in Fig. 4. A special atom $\_sol(i)$ is used to represent membership of hypothesis h_i in a solution. Rules connecting $\_sol(i)$ to h_i and constraints for the observations are added to $\mathcal{P}$.

Description of Algorithm 1: Intuitively, if $\_sol(i)$ is included in a stable model, the i-th hypothesis is assumed to hold (i.e., it is part of the diagnosis), whereas if $\_notsol(i)$ is included, the i-th hypothesis is not part of the diagnosis. Due to the minimality of stable models and because $\_sol$ and $\_notsol$ do not occur in other heads, it is impossible that both $\_sol(i)$ and $\_notsol(i)$ occur in one stable model.

The rules in instruction (4) of the function in Fig. 4 generate a correspondence between the atoms $\_sol(i)$[7] and the hypotheses h_i. If $\_sol(i)$ belongs to a model of the program, also h_i will be deduced.

The constraints generated by instructions (8) and (10) of the function in Fig. 4 ensure that any solution Δ entails, through the initial theory T, the observations (recall that Δ is an abductive diagnosis if $\Delta \cup T \models O$). $\Box$

Generic abductive diagnoses are then computed through the program which is generated by Algorithm 1. More formally:

[7] Predicate names starting with an underscore are reserved for `dlv` internal usage and cannot be used by the regular user.

Function $AbductionToDLP(\mathcal{P}) : DLP$
Input: An Abductive Problem $\mathcal{P} = \langle H, T, O \rangle$.
Output: A Datalog$^{\vee,\neg,s}$ program.
var i, j: $Integer$;
$\quad\mathcal{Q} : DLP$;
begin
$\quad\mathcal{Q} := T$;
$\quad$Let $H = \langle h_1, \ldots, h_n \rangle$;
$\quad$**for** $i := 1$ **to** n **do**
$\qquad\mathcal{Q} := \mathcal{Q} \cup \{ h_i \leftarrow \_sol(i) \}$;
$\quad$Let $O = \{ o_1, \ldots, o_m \}$;
$\quad$**for** $j := 1$ **to** m **do**
$\qquad$**if** o_j is a positive literal "a" **then**
$\qquad\quad\mathcal{Q} := \mathcal{Q} \cup \{ \leftarrow \neg\, a \}$;
$\qquad$**else** (* o_j is a negative literal "$\neg\, a$" *)
$\qquad\quad\mathcal{Q} := \mathcal{Q} \cup \{ \leftarrow a \}$;
$\quad$**return** $\mathcal{Q}$;
end

Figure 4: Basic Abductive Diagnosis Translation

Function $GenericAbductionToDLP(\mathcal{P}) : DLP$
Input: An Abductive Problem $\mathcal{P} = \langle H, T, O \rangle$.
Output: A Datalog$^{\vee,\neg,s}$ program.
var $\mathcal{Q} : DLP$;
begin
$\quad\mathcal{Q} := AbductionToDLP(\mathcal{P})$;
$\quad$Let $H = \langle h_1, \ldots, h_n \rangle$;
$\quad$**for** $i := 1$ **to** n **do**
$\qquad\mathcal{Q} := \mathcal{Q} \cup \{ \_sol(i) \vee \_notsol(i) \leftarrow \}$;
$\quad$**return** $\mathcal{Q}$;
end

Algorithm 1: Generic Abductive Diagnosis

Function $ModelToDiagnosis(M)$: *set of atoms*
Input: A Model M of a program transformed from
a diagnostic problem $\mathcal{P} = \langle H, T, O \rangle$
Output: A set of atoms (the corresponding diagnosis).
var S : *setofatoms*;
begin
(1) $S := \emptyset$;
(2) Let $H = \langle h_1, \ldots, h_n \rangle$;
(3) **for** $i := 1$ **to** n **do**
(4) **if**_*sol*$(i) \in M$ **then**
(5) $S := S \cup \{h_i\}$;
(6) **return** $\mathcal{Q}$;
end

Figure 5: Translating stable models to diagnoses

Theorem 1 *Let $\mathcal{P} = \langle H, T, O \rangle$ be an abductive problem and $DLP(\mathcal{P})$ the logic program generated by Algorithm 1 with input $\mathcal{P}$. Each stable model of $DLP(\mathcal{P})$ corresponds to precisely one diagnosis of $\mathcal{P}$, and each diagnosis of $\mathcal{P}$ has some corresponding stable model of $DLP(\mathcal{P})$. In particular, if M is a stable model of $DLP(\mathcal{P})$, then $ModelToDiagnosis(M)$ (see Fig. 5) is a* diagnosis *for $\mathcal{P}$.*
□

Demonstration: The rules from instruction (4) of Algorithm 1 generate all possible diagnosis candidates by assuming either _*sol*(i) or _*notsol*(i) for each hypothesis h_i. These candidates are then checked by the rules from Algorithm 4 as described above.
□

The program for computing a single error diagnosis is generated by Algorithm 2. It is similiar to the previous case but introduces a different choice rule.

Theorem 2 *Let $\mathcal{P} = \langle H, T, O \rangle$ be an abductive problem and $DLP(\mathcal{P})$ the logic program generated by Algorithm 2 with input $\mathcal{P}$. Each stable model of $DLP(\mathcal{P})$ corresponds to precisely one single error diagnosis of $\mathcal{P}$, and each single error diagnosis of $\mathcal{P}$ has some corresponding stable model of $DLP(\mathcal{P})$. In particular, if M is a stable model of $DLP(\mathcal{P})$, then $ModelToDiagnosis(M)$ (see Fig. 5) is a* single error diagnosis *for $\mathcal{P}$.*
□

Demonstration: This case is very similiar to the generic one (Algorithm 1 and Theorem 1), the only difference being the generation of the "guess": Instead of

Function $SingleErrorAbductionToDLP(\mathcal{P})$: DLP
Input: An Abductive Problem $\mathcal{P} = \langle H, T, O \rangle$.
Output: A Datalog$^{\vee,\neg,s}$ program.
var $\mathcal{Q}$: DLP;
begin
(1) $\mathcal{Q} := AbductionToDLP(\mathcal{P})$;
(2) Let $H = \langle h_1, \ldots, h_n \rangle$;
(3) $\mathcal{Q} := \mathcal{Q} \cup \{\_sol(1) \vee \ldots \vee \_sol(n) \leftarrow \}$;
(4) **return** $\mathcal{Q}$;
end

Algorithm 2: Single Error Abductive Diagnosis

evaluating all possible combinations of hypotheses (including the empty set), by virtue of instruction (3) in Algorithm 2 only those instances are generated (and checked) where exactly one hypothesis is assumed.

Indeed, by the minimality of stable models, one and only one of the $\_sol(i)$ – which do not appear elsewhere in the head – can be made true. □

Function $SubsetMinimalAbductionToDLP(\mathcal{P})$
: DLP
Input: An Abductive Problem $\mathcal{P} = \langle H, T, O \rangle$
where T is positive and non-disjunctive.
Output: A Datalog$^{\vee,\neg,s}$ program.
var $\mathcal{Q}$: DLP;
begin
(1) $\mathcal{Q} := GenericAbductionToDLP(\mathcal{P})$;
(2) $\mathcal{Q} := \mathcal{Q} \cup AbductiveSubsetMinimality(\mathcal{P})$;
(3) **return** $\mathcal{Q}$;
end

Algorithm 3: Subset Minimal Abductive Diagnosis

The program for computing subset minimal abductive diagnoses is generated by Algorithm 3. Basically, it consists of a part which generates a solution, and a part which checks for minimality. This part is produced by the function in Fig. 6, which is the most complicated function considered here.

Note that the theory is restricted to be nondisjunctive and nonnegative because the full subset minimal abductive diagnosis problem over Datalog$^{\vee,\neg,s}$ theories is in

Function $AbductiveSubsetMinimality(\mathcal{P}) : DLP$
Input: An Abductive Problem $\mathcal{P} = \langle H, T, O \rangle$
where T is positive and non-disjunctive.
Output: A Datalog$^{\vee,\neg,s}$ program.
var i, j: $Integer$;
$\mathcal{Q}$: DLP;
begin
(1) $\mathcal{Q} := \emptyset$;
(2) Let $H = \langle h_1, \ldots, h_n \rangle$;
(3) **for** $i := 1$ **to** n **do**
(4) $\mathcal{Q} := \mathcal{Q} \cup \{\_hyp(i).\}$;
(5) $\mathcal{Q} := \mathcal{Q} \cup \{\_sol1(K, I) \leftarrow \_sol(K), \_hyp(I),$
$<> (K, I).\}$;
(6) **for** $i := 1$ **to** n **do**
(7) $\mathcal{Q} := \mathcal{Q} \cup \{\_\_h_i(\overline{x}, I) \leftarrow \_sol1(i, I)\ \}$;
(8) **for** each $r : a(\overline{x}) \leftarrow b_1(\overline{x_1}), \ldots, b_n(\overline{x_n})\ \in T$ **do**
(9) $\mathcal{Q} := \mathcal{Q} \cup \{\_\_a(\overline{x}, I) \leftarrow \_hyp(I)\ ,$
$\_\_b_1(\overline{x_1}, I), \ldots, \_\_b_n(\overline{x_n}, I)\}$;
(10) Let $O = \{o_1, \ldots, o_m\}$;
(11) **for** $j := 1$ **to** m **do**
(12) $\mathcal{Q} := \mathcal{Q} \cup \{\_notobs(I) \leftarrow \_hyp(I), \neg\ \_\_o_j(\overline{x}, I)\ \}$;
(13) $\mathcal{Q} := \mathcal{Q} \cup \{\_isntsol(0) \leftarrow \}$;
(14) $\mathcal{Q} := \mathcal{Q} \cup \{\_isntsol(I) \leftarrow \_isntsol(K),$
$\#succ(K, I), \_notobs(I).\}$;
(15) $\mathcal{Q} := \mathcal{Q} \cup \{\_isntsol(I) \leftarrow \_isntsol(K),$
$\#succ(K, I), \neg\ \_sol(I).\}$;
(16) $\mathcal{Q} := \mathcal{Q} \cup \{\leftarrow \neg\ \_isntsol(n).\ \}$;
(17) **return** $\mathcal{Q}$;
end;

Figure 6: Subset Minimality Check for Abductive Diagnosis

the complexity class Σ_3^P, and thus not expressible in Datalog$^{\vee,\neg,s}$ (cf. [11]). Subset minimal abductive diagnosis over Datalog$^{\neg}$ programs would be expressible, but is not considered in order to keep the algorithm compact.

Theorem 3 *Let $\mathcal{P} = \langle H, T, O\rangle$ be an abductive problem and $DLP(\mathcal{P})$ the logic program generated by Algorithm 3 with input $\mathcal{P}$. Each stable model of $DLP(\mathcal{P})$ corresponds to precisely one subset minimal diagnosis of $\mathcal{P}$, and each subset minimal diagnosis of $\mathcal{P}$ has some corresponding stable model of $DLP(\mathcal{P})$. In particular, if M is a stable model of $DLP(\mathcal{P})$, then $ModelToDiagnosis(M)$ (see Fig. 5) is a* subset minimal diagnosis *for $\mathcal{P}$.*
$\square$

Demonstration: First of all note that $<>$ and $\#succ$ are built-in predicates directly supported by `dlv`, where $<>$ stands for inequality and $\#succ(A, B)$ denotes $A+1 = B$, i.e., "B is the successor of A".

To compute minimal diagnoses, we extend the generic case (Algorithm 1) with an additional minimality check that is performed by a set of rules generated by the function *AbductiveSubsetMinimality* (depicted in Fig. 6).

Basically, this minimality check works as follows: Given a diagnosis Δ (which is a subset of the hypotheses satisfying the constraints) with n elements, all the subsets of Δ having $n-1$ elements are generated. If one of these subsets itself is a diagnosis, then Δ is rejected.

The rules from instruction (4) of the function in Fig. 6 generate a set of facts $\_hyp(1), \ldots, \_hyp(n)$ that will be used to enumerate all hypotheses later on.

The rule from instruction (5) generates the subsets of Δ as follows: For any fixed index I $(1 \leq I \leq n)$, $\Delta_I = \{\_sol1(K, I) : h_K \in \Delta, K \neq I\}$ represents the set of all elements of Δ except the one corresponding to the I-th hypothesis h_I. In other words, $\_sol1(K, I)$ is true in some stable model M if and only if $\_sol(K)$ is true in M and $I \neq K$.

For each of these subsets Δ_I of Δ, we need to check whether it is a diagnosis, that is, whether Δ_I entails the observations. If so, we can reject Δ because it is not minimal. By means of instructions (8) and (9) we generate a new version of the theory T parametrized with respect to the index I of Δ_I.

The rules generated by instruction (7) install the correspondence between the elements of Δ_I (stored in $\_sol1(K, I)$) and the hypotheses parametrized with respect to I (i.e., $h(\overline{x})$ becomes $\_\_h(\overline{x}, I)$, where $\overline{x}$ stands for the parameter list of h.)

The rules from instruction (12) check whether the observations are really entailed by Δ_I by means of the parametrized copy of the theory T that has been generated by instructions (8) and (9). If an observation is not entailed by Δ_I, then $\_notobs(I)$ becomes true. This means that Δ_I is not a diagnosis and it thus does not invalidate the minimality of Δ.

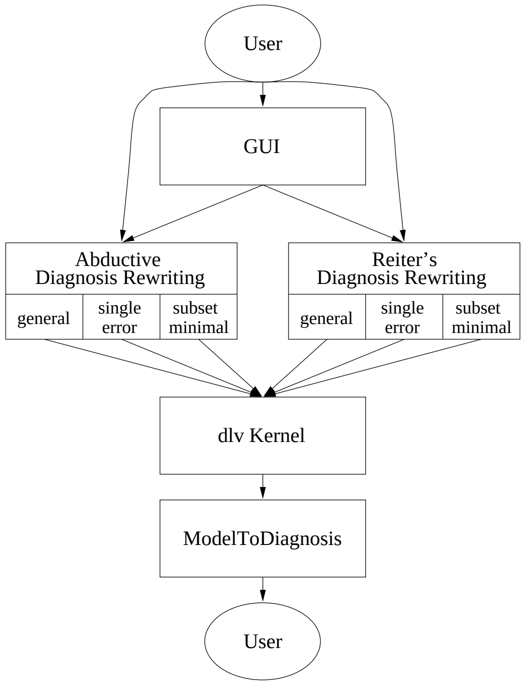

Figure 7: System Architecture

If all the subsets Δ_I of Δ verify this condition (that is, $\_notobs(I)$ is true for every I) then Δ is indeed a minimal diagnosis, otherwise it is rejected.

This final check is performed by the rules from instructions (13), (14), (15) and (16): If Δ is a minimal solution, then for each $K \in \{1, \ldots, n\}$, it must hold that either (*) $h_K \in \Delta$ and Δ_K is not a solution, or (**) $h_K \notin \Delta$. Thus, for each K, $\_isnotsol(K)$ must be true.

The conditions (*) and (**) are expressed by the rules (14) and (15), respectively. They process the hypotheses along their ordering, and are only applicable if the checks for all preceeding hypotheses were successful. Rule (13) is the initialization, and rule (16) enforces that the check is true for all hypotheses. □

6 System Architecture

The general architecture of the diagnosis frontend of `dlv` is depicted in Fig. 7.

In order to utilise the diagnosis frontend, the user has to keep the theory, the hypotheses and the observations of a diagnostic problem in separate files. The hypotheses file is required to bear the extension "`.hyp`", the observations file must be tagged with the extension "`.obs`", while for the theory file there is no special

Option	Effect
-FD	generic abductive diagnoses
-FDsingle	single error abductive diagnoses
-FDmin	subset minimal abductive diagnoses
-FR	“Reiter's” (consistency-based) diagnoses
-FRsingle	single error “Reiter's” diagnoses
-FRmin	subset minimal “Reiter's” diagnoses

Table 1: Command-line switches

requirement[8].

For invoking the `dlv` system on a diagnosis problem, a traditional command-line interface and a *graphical user interface* (GUI) are available to the user.

The type of reasoning (abductive or consistency-based) and the desired minimality criterion (none, single error, subset minimality) is selected by specifying one of the options depicted in Table 1 on the command-line or by means of a radio button in the GUI.

Depending on these diagnosis mode settings, the corresponding translation algorithms are applied, as indicated in Fig. 7. The resulting Datalog$^{\vee,\neg,s}$ program is then solved by `dlv`, i.e., all stable models are calculated.

Finally, the function $ModelToDiagnosis$ (see Fig. 5) is applied to each model which has been found, in order to extract the diagnosis which corresponds to that model. The user interface subsequently displays the diagnoses in a window or on the terminal.

7 Conclusion and Further Work

We have described the diagnosis frontend of the `dlv` system. It allows for both abductive and consistency-based diagnostic reasoning and, for each diagnosis modality, generic diagnoses, single-error diagnoses, and subset minimal diagnoses can be computed.

The knowledge representation power of the formalism has been illustrated by showing a number of problems which can be represented very easily in our framework. The implementation of the frontend relies on translation algorithms, which rewrite diagnosis problems into disjunctive logic programs whose stable models correspond to diagnoses. Some of these algorithms, and in particular those for subset minimal diagnosis, make use of novel DLP encoding techniques, which fully exploit the power of disjunctive logic programming to solve Σ_2^P-complete problems.

[8]However, we recommend using “`.dl`” or no extension at all to avoid confusion.

Several issues remain for further work. One issue is the design of optimization techniques that speed up the computation of diagnoses.[9] Note that a datalog program per se can not be considered more or less efficient. However, fragments of Datalog$^{\vee,\neg,s}$ can be identified for which a lower complexity bound can be guaranteed and which therefore allow potentially more efficient solution algorithms.

Thus, exploiting structural knowledge about the systems to be diagnosed, e.g., as in [7, 8, 36], is suggestive and may lead to increased performance. This requires more sophisticated, problem specific translation algorithms mapping diagnosis problems into disjunctive logic programs contained in a fragment of Datalog$^{\vee,\neg,s}$, for which more efficient evaluation procedures are known. The datalog engine then might recognise these restrictions and employ less complex algorithms to find solutions. However, all these optimisations would be transparent to the user.

In this paper, Datalog$^{\vee,\neg,s}$ has been presented as a unifying computational framework for different abductive problems. The identification of a unifying semantical framework for abduction, using a generalized definition of diagnosis problems in the spirit of [6], remains for further investigation.

Another issue for further work is the inclusion of alternative formalizations of diagnosis into the front-end, e.g. abductive diagnosis under inference for all stable models.

The `dlv` system is freely available for various platforms via the `dlv` homepage [18].

More detailed descriptions of other features of the `dlv` system can be found in [1, 4, 13, 14, 16, 15, 17].

Acknowledgements

The authors wish to thank Simona Citrigno for implementing the original version of the diagnosis frontend of `dlv`.

[9]Currently we cannot expect very fast computations, as our diagnosis frontend solves Σ_2^P-hard problems.

Appendix

A Consistency-Based Diagnosis

In this appendix we present the algorithms that rewrite CDPs (as defined in Section 4) to disjunctive logic programs (as defined in 2). As in the case of ADPs, we will first give the algorithm for generic consistency-based diagnoses, then for single error and finally for subset minimal diagnoses.

For consistency based problems, we impose some syntactic restrictions: As stated after Definition 7, hypotheses should consist only of the predicate `ab`, and negative literals in the body are only allowed if the involved atoms are hypotheses.

Additionally, for subset minimal diagnostic problems we do not allow disjunctions and hypotheses may only occur negatively (i.e. as `not ab(...)`).

The basic consistency-based translation is performed by the function in Fig. 8, which is similar to the basic transformation in the abductive case (see Fig. 4). The difference is that positive observations `a` are translated to facts `a` $\leftarrow$ instead of constraints $\leftarrow \neg$ `a` (lines (10) and (12) in the function in Fig. 8).

$\leftarrow \neg$ `a` is introduced in the abductive case to disallow models in which the observation `a` does not hold. In the consistency-based case positive observations need not be entailed. Rather, if we assume them to hold (by adding the corresponding fact to the theory), they must not give reason to inconsistencies. Negative observations need not be inserted as facts, because of the concept of negation as failure. Rather, we must ensure that the positive variants of negative observations are not derived (since this would represent an inconsistency). Therefore we add $\leftarrow \neg$ `a` .

Additionally we have to include constraints for each hypothesis, which forbids the hypothesis to become true unless the corresponding `_sol(i)` is also true (line (6)). If `_sol(i)` does not hold, the corresponding hypothesis is assumed to be false (cf. Definition 8). If the same hypothesis is then derived (i.e. becomes true), the diagnosis leads to an inconsistency. These constraints could be omitted if we disallowed hypotheses to occur in the head of rules in the theory, because then the only way in which hypotheses could become true is via the corresponding `_sol(i)`.

Theorem 4 *Let $\mathcal{P} = \langle H, T, O \rangle$ be a consistency-based problem and $DLP(\mathcal{P})$ the logic program generated by Algorithm 4 with input $\mathcal{P}$. There exists a many to one correspondence between the stable models of $DLP(\mathcal{P})$ and the diagnoses of $\mathcal{P}$. In particular, if M is a stable model of $DLP(\mathcal{P})$, then $ModelToDiagnosis(M)$ (see Fig. 5) is a* diagnosis *for $\mathcal{P}$.*
□

Demonstration: As in Algorithm 1, the rules in instruction (4) of Algorithm 4 generate all possible diagnosis candidates by assuming either $\_sol(i)$ or $\_notsol(i)$ for

Function $ConsBasedToDLP(\mathcal{P})$: DLP
Input: A Consistency Based Problem $\mathcal{P} = \langle H, T, O \rangle$
where T is positive (except for hypotheses) and non-disjunctive.
Output: A Datalog$^{\vee,\neg,s}$ program.
var i, j: $Integer$;
$\mathcal{Q}$: DLP;
begin
$\mathcal{Q} := T$;

Let $H = \langle h_1, \ldots, h_n \rangle$;
for $i := 1$ **to** n **do**
$\mathcal{Q} := \mathcal{Q} \cup \{ h_i \leftarrow \_sol(i) \}$;
for $i := 1$ **to** n **do**
$\mathcal{Q} := \mathcal{Q} \cup \{ \leftarrow h_i, \neg \_sol(i) \}$;

Let $O = \{o_1, \ldots, o_m\}$;
for $j := 1$ **to** m **do**
if o_j is a positive literal "a" **then**
$\mathcal{Q} := \mathcal{Q} \cup \{ a \leftarrow \}$;
else ($*$ o_j is a negative literal "$\neg a$" $*$)
$\mathcal{Q} := \mathcal{Q} \cup \{ \leftarrow a \}$;

return $\mathcal{Q}$;
end

Figure 8: Basic Consistency Based Diagnosis Translation

Function $GenericConsBasedToDLP(\mathcal{P})$: DLP
Input: A Consistency Based Problem $\mathcal{P} = \langle H, T, O \rangle$
where T is positive (except for hypotheses) and non-disjunctive.
Output: A Datalog$^{\vee,\neg,s}$ program.
var $\mathcal{Q}$: DLP;
begin
(1) $\mathcal{Q} := ConsBasedToDLP(\mathcal{P})$;
(2) Let $H = \langle h_1, \ldots, h_n \rangle$;
(3) **for** $i := 1$ **to** n **do**
(4) $\mathcal{Q} := \mathcal{Q} \cup \{\_sol(i) \vee \_notsol(i) \leftarrow \}$;
(5) **return** $\mathcal{Q}$;
end

Algorithm 4: Generic Consistency-Based Diagnosis

each hypothesis h_i. These candidates are then checked by the rules from the function in Fig. 8, as described above. □

Function $SingleErrorConsBasedToDLP(\mathcal{P})$: DLP
Input: A Consistency Based Problem $\mathcal{P} = \langle H, T, O \rangle$
where T is positive (except for hypotheses) and non-disjunctive.
Output: A Datalog$^{\vee,\neg,s}$ program.
var $\mathcal{Q}$: DLP;
begin
(1) $\mathcal{Q} := ConsBasedToDLP(\mathcal{P})$;
(2) Let $H = \langle h_1, \ldots, h_n \rangle$;
(3) $\mathcal{Q} := \mathcal{Q} \cup \{\_sol(1) \vee \ldots \vee \_sol(n) \leftarrow \}$;
(4) **return** $\mathcal{Q}$;
end

Algorithm 5: Single Error Consistency Based Diagnosis

Single error diagnoses of CDPs are translated in analogy to Algorithm 2. The only difference is that the basic translation is accomplished by the function $ConsBasedToDLP$ (see Fig. 8).

Theorem 5 *Let $\mathcal{P} = \langle H, T, O \rangle$ be a consistency-based diagnostic problem and $DLP(\mathcal{P})$ the logic program generated by Algorithm 5 with input $\mathcal{P}$. There exists a many to*

one correspondence between the stable models of $DLP(\mathcal{P})$ and a single error diagnosis of $\mathcal{P}$ and vice versa. In particular, if M is a stable model of $DLP(\mathcal{P})$, then $ModelToDiagnosis(M)$ is a single error diagnosis *for $\mathcal{P}$.*
$\Box$

Demonstration: This case is very similiar to the generic one (Algorithm 4 and Theorem 4), and as in Algorithm 2, line (3) ensures that exactly one hypothesis can become true because of the minimality of stable models. $\Box$

Function $SubsetMinimalConsBasedToDLP(\mathcal{P})$
: DLP
Input: A Consistency Based Problem $\mathcal{P} = \langle H, T, O\rangle$
where T is positive (except for hypotheses) and non-disjunctive.
Output: A Datalog$^{\vee,\neg,s}$ program.
var $\mathcal{Q}$: DLP;
begin
(1) $\mathcal{Q} := GenericConsBasedToDLP(\mathcal{P})$;
(2) $\mathcal{Q} := \mathcal{Q} \cup ConsBasedSubsetMinimality(\mathcal{P})$;
(3) **return** $\mathcal{Q}$;
end

Algorithm 6: Subset Minimal Consistency Based Diagnosis

The program for computing subset minimal consistency-based diagnoses is generated by Algorithm 6. As Algorithm 3, it consists of a part which generates a solution, and a part which checks for minimality. This part is produced by the function in Fig. 9, which in turn is similar to the one depicted in Fig. 6.

Theorem 6 *Let $\mathcal{P} = \langle H, T, O\rangle$ be a consistency-based diagnostic problem and $DLP(\mathcal{P})$ the logic program given by Algorithm 6 with input $\mathcal{P}$. There exists a many to one correspondence between a stable models of $DLP(\mathcal{P})$ and the subset minimal diagnoses of $\mathcal{P}$ and vice versa. In particular, if M is a stable model of $DLP(\mathcal{P})$, then $ModelToDiagnosis(M)$ (see Fig. 5) is a* subset minimal diagnosis *for $\mathcal{P}$.*
$\Box$

Demonstration: The algorithm is based on the same ideas as its abductive counterpart, as demonstrated after Theorem 3. Only minor changes have to be applied to the function which generates the minimality check (depicted in Fig. 6), which stem from the different definition of diagnoses and from the different syntactical restrictions.

The differences are how the observations are translated (lines (14) to (17)) and that constraints (and not only rules) have to be parametrized w.r.t. hypotheses of a diagnosis which has to be checked for subset minimality (lines (10) and (11)).

Function $ConsBasedSubsetMinimality(\mathcal{P}) : DLP$
Input: A Consistency Based Problem $\mathcal{P} = \langle H, T, O \rangle$
where T is positive (except for hypotheses) and non-disjunctive,
and hypotheses may only occur as negative literals.
Output: A Datalog$^{\vee,\neg,s}$ program.
var i, j: $Integer$;
$\mathcal{Q}$: DLP;
begin
(1) $\mathcal{Q} := \emptyset$;
(2) Let $H = \langle h_1, \ldots, h_n \rangle$;
(3) **for** $i := 1$ **to** n **do**
(4) $\quad \mathcal{Q} := \mathcal{Q} \cup \{\_hyp(i)\}$;
(5) $\mathcal{Q} := \mathcal{Q} \cup \{\_sol1(K, I) \leftarrow \_sol(K), \_hyp(I), <> (K, I)\ \}$;
(6) **for** $i := 1$ **to** n **do**
(7) $\quad \mathcal{Q} := \mathcal{Q} \cup \{\_\_h_i(\overline{x}, I) \leftarrow \_sol1(i, I)\ \}$;
(8) **for** each $r : a(\overline{x}) \leftarrow b_1(\overline{x_1}), \ldots, b_n(\overline{x_n})\ \in T$ **do**
(9) $\quad \mathcal{Q} := \mathcal{Q} \cup \{\_\_a(\overline{x}, I) \leftarrow \_hyp(I), \_\_b_1(\overline{x_1}, I), \ldots, \_\_b_n(\overline{x_n}, I)\ \}$;
(10) **for** each $c :\leftarrow c_1(\overline{x_1}), \ldots, c_n(\overline{x_n})\ \in T$ **do**
(11) $\quad \mathcal{Q} := \mathcal{Q} \cup \{\_notobs(I) \leftarrow \_\_c_1(\overline{x_1}, I), \ldots, \_\_c_n(\overline{x_n}, I), \_hyp(I)\ \}$;
(12) Let $O = \{o_1, \ldots, o_m\}$;
(13) **for** $j := 1$ **to** m **do**
(14) $\quad$ **if** o_j is a positive literal "$a(\overline{x})$" **then**
(15) $\qquad \mathcal{Q} := \mathcal{Q} \cup \{a(\overline{x}, I) \leftarrow \_hyp(I)\ \}$;
(16) $\quad$ **else** (* o_j is a negative literal "$\neg\, a(\overline{x})$" *)
(17) $\qquad \mathcal{Q} := \mathcal{Q} \cup \{\_notobs(I) \leftarrow \_hyp(I), a(\overline{x}, I)\ \}$;
(18) $\mathcal{Q} := \mathcal{Q} \cup \{\_isntsol(0) \leftarrow\ \}$;
(19) $\mathcal{Q} := \mathcal{Q} \cup \{\_isntsol(I) \leftarrow \_isntsol(K),$
$\#succ(K, I), \_notobs(I)\}$;
(20) $\mathcal{Q} := \mathcal{Q} \cup \{\_isntsol(I) \leftarrow \_isntsol(K),$
$\#succ(K, I), \neg\, \_sol(I)\}$;
(21) $\mathcal{Q} := \mathcal{Q} \cup \{\leftarrow \neg\, \_isntsol(n)\ \}$;
(22) **return** $\mathcal{Q}$;
end;

Figure 9: Subset Minimality Check for Consistency Based Diagnosis

The observations are translated in analogy to the basic consistency-based translation in Fig. 8.

The reason why constraints must be parametrised in the consistency-based minimality check, but not in the abductive one is due to the restrictions which are imposed on the theories:

In the abductive case, we only allow positive, non-disjunctive theories. If some hypothesis is omitted from a diagnosis, no additional atoms can be derived (because the program is positive and non-disjunctive). Therefore, if a constraint was satisfied for a solution, at least one of its atoms[10] was false, and it must also be false for the smaller diagnosis candidate.

In the consistency-based case, hypotheses may occur negatively in bodies. For this reason, if a hypothesis is omitted from a valid diagnosis, a formerly satisfied constraint may become violated (for example, if the hypothesis which is omitted from the diagnosis occurs negatively in a constraint), so parametrized constraints must be included. These constraints are rewritten to rules — if one of their bodies becomes true, an atom is derived which represents the fact that this subdiagnosis is not a valid diagnosis (`_notobs(I)`).
□

References

[1] R. Bihlmeyer, W. Faber, C. Koch, N. Leone, C. Mateis, and G. Pfeifer. `dlv` – an overview. In *Proceedings of the 13th Workshop on Logic Programming (WLP '98)*, October 1998.

[2] F. Buccafurri, N. Leone, and P. Rullo. Strong and Weak Constraints in Disjunctive Datalog. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR '97)*, number 1265 in Lecture Notes in AI (LNAI), pages 2–17, Dagstuhl, Germany, July 1997. Springer Verlag.

[3] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer, 1990.

[4] S. Citrigno, T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The `dlv` System: Model Generator and Application Frontends. In F. Bry, B. Freitag, and D. Seipel, editors, *Proceedings of the 12th Workshop on Logic Programming (WLP '97), Research Report PMS-FB10*, pages 128–137, München, Germany, September 1997. LMU München.

[10] Since only positive theories are allowed, a constraint contains only atoms.

[5] L. Console, D. Theseider Dupré, and P. Torasso. On the Relationship Between Abduction and Deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

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

[7] A. Darwiche. Model-Based Diagnosis using Causal Networks. In C. S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 211–217. Morgan Kaufmann Publishers, 1995.

[8] A. Darwiche. Compiling Devices: A Structure-Based Approach. In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 156–166. Morgan Kaufmann Publishers, 1998.

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

[10] J. Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In *Logic, Action and Information. Proc. of the Konstanz Colloquium in Logic and Information (LogIn'92)*, pages 241–329. DeGruyter, 1995.

[11] T. Eiter, G. Gottlob, and N. Leone. Abduction from Logic Programs: Semantics and Complexity. *Theoretical Computer Science*, 189(1–2):129–177, December 1997.

[12] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):315–363, September 1997.

[13] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Nonmonotonic Reasoning. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '97)*, number 1265 in Lecture Notes in AI (LNAI), pages 363–374, Berlin, 1997. Springer.

[14] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The Architecture of a Disjunctive Deductive Database System. In M. Falaschi, M. Navarro, and A. Policriti, editors, *Proceedings Joint Conference on Declarative Programming (APPIA-GULP-PRODE '97)*, pages 141–151, June 1997.

[15] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. Progress Report on the Disjunctive Deductive Database System `dlv`. In T. Andreasen, H. Christiansen, and H. L. Larsen, editors, *Proceedings International Conference on Flexible Query Answering Systems (FQAS '98)*, pages 145–160, Roskilde University, Denmark, May 1998.

[16] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System `dlv`: Progress Report, Comparisons and Benchmarks. In A. G. Cohn, L. Schubert, and

S. C. Shapiro, editors, *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 406–417. Morgan Kaufmann Publishers, 1998.

[17] W. Faber, N. Leone, and G. Pfeifer. Representing school timetabling in a disjunctive logic programming language. In *Proceedings of the 13th Workshop on Logic Programming (WLP '98)*, October 1998.

[18] W. Faber and G. Pfeifer. `dlv` homepage. `http://www.dbai.tuwien.ac.at/proj/dlv/`.

[19] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[20] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[21] K. Inoue and C. Sakama. Transforming Abductive Logic Programs to Disjunctive Programs. In D. S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 335–353, 1993.

[22] A. Kakas, R. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 1993.

[23] A. Kakas and P. Mancarella. Database Updates Through Abduction. In *Proceedings of the 16th VLDB Conference*, pages 650–661, Brisbane, Australia, 1990.

[24] K. Konolige. Abduction versus closure in causal theories. *Artificial Intelligence*, 53:255–272, 1992.

[25] N. Leone, P. Rullo, and F. Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation*, 135(2):69–112, June 1997.

[26] J. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, 1984.

[27] J. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, 1987. second edition.

[28] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* The MIT Press, Cambridge, Massachusetts, 1992.

[29] J. Minker. On Indefinite Data Bases and the Closed World Assumption. In D. Loveland, editor, *Proceedings* 6^{th} *Conference on Automated Deduction (CADE '82)*, number 138 in Lecture Notes in Computer Science, pages 292–308, New York, 1982. Springer.

[30] Y. Peng and J. Reggia. *Abductive Inference Models for Diagnostic Problem Solving.* Symbolic Computation – Artificial Intelligence. Springer, 1990.

[31] D. Poole. Explanation and Prediction: An Architecture for Default and Abductive Reasoning. *Computational Intelligence*, 5(1):97–110, 1989.

[32] D. Poole. Normality and Faults in Logic-Based Diagnosis. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1304–1310, Detroit, Michigan, USA, 1989.

[33] T. C. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9:401–424, 1991.

[34] R. Reiter. A Theory of Diagnosis From First Principles. *Artificial Intelligence*, 32:57–95, 1987.

[35] C. Sakama and K. Inoue. On the Equivalence between Disjunctive and Abductive Logic Programs. In P. Van Hentenryck, editor, *Proceedings of the Eleventh International Conference on Logic Programming*, pages 489–503, 1994. Extended version "On the Relation between Abductive and Disjunctive Logic Programming" (1998).

[36] M. Stumptner and F. Wotawa. Diagnosing tree-structured systems. In *Proceedings 15th International Joint Conference on Artificial Intelligence*, pages 440–445, Nagoya, Japan, 1997.