Model-Based Diagnosis with Constraint Logic Programs[*]

Igor Mozetič
Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A-1010 Vienna, Austria

Christian Holzbaur
Austrian Research Institute for Artificial Intelligence, and
Department of Medical Cybernetics and Artificial Intelligence
University of Vienna
Freyung 6, A-1010 Vienna, Austria

Abstract

Model-based diagnosis is the activity of locating malfunctioning components of a system solely on the basis of its structure and behavior. In the paper we describe the role of Constraint Logic Programming (CLP) in representing models and the search space of minimal diagnoses. In particular, we concentrate on two instances of the CLP scheme: CLP($\mathcal{B}$) and CLP($\Re$). CLP($\mathcal{B}$) extends the standard computational domain of logic programs by boolean expressions, while CLP($\Re$) comprises a solver for systems of linear equations and inequalities over real-valued variables.

1 Introduction

There are two fundamentaly different approaches to diagnostic reasoning. In the first, heuristic approach, one encodes diagnostic rules of thumb and experience of human experts in a given domain. In the second, model-based approach, one starts with a model of a real-world system which explicitly represents the structure and components of the system (e.g., Genesereth 1984, Davis 1984, de Kleer & Williams 1987, Reiter 1987). When the system's

[*]Appears in *Proc. 7th Austrian Conf. on Artificial Intelligence, ÖGAI-91* (H. Kaindl, Ed.), pp. 168-180, Vienna, Austria, September 25-27, 1991, Springer-Verlag (IFB 287).

actual behavior is different from the expected behavior, the diagnostic problem arises. The model is then used to identify faulty components and their internal states which account for the observed behavior. One usually requires a parsimonious diagnosis, i.e., a *minimal* set of faulty components.

In our view there are two major obstacles which prevented a wider application of model-based techniques to real-world problems. First, the complexity of algorithms which find all minimal diagnoses is exponential in the number of components. This problem was addressed only recently, either by focusing just to a small number of most probable diagnoses (de Kleer & Williams 1990, Mozetic & Holzbaur 1991b), by interleaving diagnosis and treatment (Friedrich *et al.* 1990), or by introducing abstractions (Gallanti *et al.* 1989, Mozetic 1990). Second, models are usually restricted to qualitative descriptions. GDE (de Kleer & Williams 1987), for example, is unable to solve simultaneous equations, which makes it unpractical for a large class of applications.

In the paper we describe the role of Constraint Logic Programming (CLP, Jaffar *et al.* 1986, Cohen 1990) in representing a larger class of models. CLP are logic programs extended by interpreted functions. A proper implementation of the CLP scheme allows for an easy integration of specialized problem solvers into the logic programming framework. For example, in Metaprolog (an extension of C-Prolog, Holzbaur 1990) specialized solvers communicate with the standard Prolog interpreter via extended semantic unification and are implemented in Prolog themselves. So far, we have implemented three solvers: constraint propagation over finite domains by forward checking, CLP($\mathcal{B}$) — a solver over boolean expressions, and CLP($\Re$) — a solver for systems of linear equations and inequalities over reals[1].

In section 2 we show how to model structure and behavior by logic programs. We clarify the relationship between the behavior of a component and its internal state (normal or malfunctioning). In section 3 we define the concepts of a diagnosis and conflict and show how to compute them by a logic programming system. For compact modeling of discrete systems, e.g., digital circuits, and for representing the search space of minimal diagnoses CLP($\mathcal{B}$) can be used (section 4). In contrast, to model continuos systems, e.g., analogue circuits, a more expressive formalism is needed. In section 5 we outline the use and operation of CLP($\Re$).

2 Modeling structure and behavior

Model-based reasoning about a system requires an explicit representation (a model) of the system's components and their connections. Reasoning is typically based on theorem proving if a model is represented by first-order logic (Genesereth 1984, Reiter 1987), or on

[1]Metaprolog and the three specialized solvers are available on request from the second author.

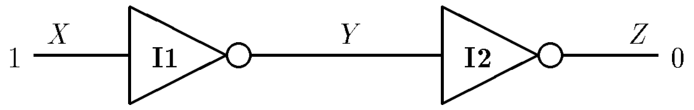

Figure 1: Two inverters, with states I1 and I2, and an observation ⟨1,0⟩.

constraint propagation coupled with an ATMS (de Kleer & Williams 1987). We represent models by *logic programs* (Lloyd 1987), or by *constraint logic programs* (Jaffar *et al.* 1986).

Definition. A *model* of a system is a triple ⟨*SD, COMPS, OBS*⟩ where

1. *SD*, the system description, is a logic program with a distinguished top-level binary predicate *m(COMPS, OBS)*.
2. *COMPS*, states of the system components, is an n-tuple $\langle S_1, \ldots, S_n \rangle$ where n is the number of components, and variables S_i denote states (e.g., normal or abnormal) of components.
3. *OBS*, observations, is an m-tuple $\langle In_1, \ldots, In_i, Out_{i+1}, \ldots, Out_m \rangle$ where *In* and *Out* denote inputs and outputs of the model, respectively.

In a logic program, n-tuples are represented by terms of arity n. Variables start with capitals and are implicitly universally quantified in front of a clause, and constants start with lower-case letters. In *SD* and definitions we refer to a distinguished constant *ok* to denote that the state S_i of the component i is normal. This corresponds to the statement $\neg ab(S_i)$ used in the consistency-based approach.

Example (two inverters, de Kleer *et al.* 1990, Figure 1). *COMPS* is a pair ⟨*I1,I2*⟩ and *OBS* is a pair ⟨*X,Z*⟩. *SD* consists of the following clause which specifies the structure of the device, and of additional clauses which define behavior of an inverter:

```
m( ⟨I1,I2⟩, ⟨X,Z⟩ )  ←
        inv( I1, X, Y ),
        inv( I2, Y, Z ).
```

Connections between components are represented by shared variables. Specification of the behavior depends on the available knowledge about possible faults. We distinguish between three types of fault models: weak, exoneration, or strong.

A **weak** fault model defines just normal behavior of components (state *ok*), abnormal behavior (state *ab*) is unconstrained:

```
inv( ok, 0, 1 ).
inv( ok, 1, 0 ).
inv( ab, _, _ ).
```

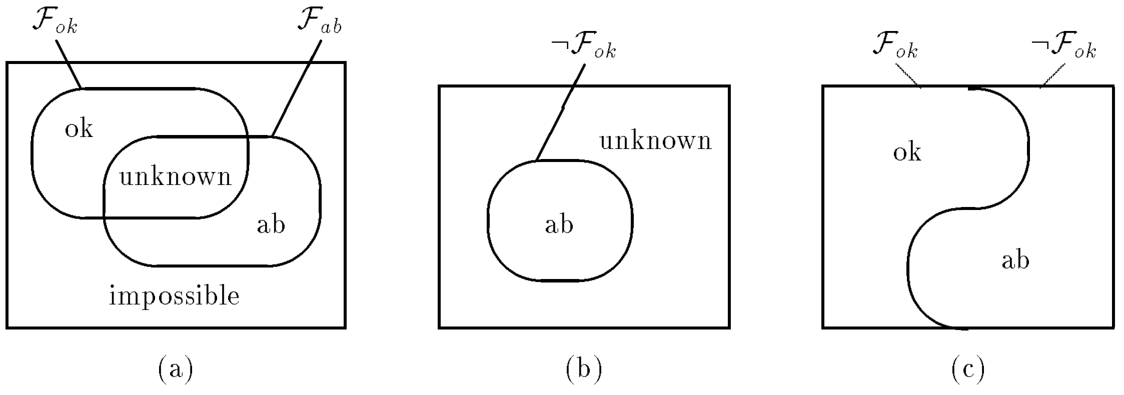

Figure 2: General relation (a) between the behavior ($\mathcal{F}_{ok}$, $\mathcal{F}_{ab}$) and the state (ok, ab, unknown) of a component. Two special cases are a weak fault model (b), and an exoneration model (c).

$\mathcal{F}_{ok}$	$\mathcal{F}_{ab}$	S	Meaning
0	0	fail	impossible
0	1	1	ab
1	0	0	ok
1	1	W	unknown

Table 1: A decision table encoding the state S of a component as a function of normal ($\mathcal{F}_{ok}$) and abnormal ($\mathcal{F}_{ab}$) behavior. $\mathcal{F} = 1$ denotes that the behavior is achieved, while $\mathcal{F} = 0$ denotes that the behavior is not achieved.

An **exoneration** model (Raiman 1989) is a special case of a strong fault model and specifies as abnormal any behavior different than normal:

inv(ok, 0, 1).
inv(ok, 1, 0).
inv(ab, X, X).

A **strong** fault model (Struss & Dressler 1989) specifies all the possible ways in which a component can fail. In general, a component may have several failure states (e.g., stuck-at-0 or stuck-at-1), but in our example we allow just for one:

inv(ok, 0, 1).
inv(ok, 1, 0).
inv(ab, _, 0).   *% stuck-at-0*

General relation between the behavior and the state of a component is depicted in Figure 2. Instead of using essentially extensional descriptions of the weak, exoneration, and strong fault models we can formulate them in terms of boolean algebra expressions. In the following $\mathcal{F}_{ok}$ denotes a normal, and $\mathcal{F}_{ab}$ an abnormal behavior of a component. If we encode the normal state as 0 (zero) instead of *ok*, and the faulty state as 1 instead of *ab* then the following expression computes the state from the behavior (Table 1):

$S \equiv (\neg\mathcal{F}_{ok} \ \vee \ \mathcal{F}_{ab} \wedge W) \quad \leftarrow \quad \mathcal{F}_{ok} \vee \mathcal{F}_{ab}.$

In a special case of a weak fault model (where $\mathcal{F}_{ab}$ is unconstrained) the state S is computed as $S \equiv (\neg\mathcal{F}_{ok} \ \vee \ W)$. If a normal behavior is not achieved we conclude that the component is faulty. Otherwise we cannot conclude anything (an unbound variable W).

In a special case of an exoneration model (where $\mathcal{F}_{ab} \equiv \neg\mathcal{F}_{ok}$) the state is simply $S \equiv \neg\mathcal{F}_{ok}$. If a normal behavior is not achieved we conclude that the component is faulty. Otherwise we conclude it functions correctly.

For the inverter example, the normal behavior is defined as $\mathcal{F}_{ok}$: ¬X ≡ Y. The following three clauses specify the weak, exoneration, and strong fault model, respectively, and $\oplus$ is the *xor* operator:

$inv(\ (\neg X \oplus\ Y) \vee\ W,\ X,\ Y\ ).$
$inv(\ \neg X \oplus\ Y,\ X,\ Y\ ).$
$inv(\ (\neg X \oplus\ Y) \vee (\neg Y \wedge\ W),\ X,\ Y\ ) \ \leftarrow\ ((\neg X \equiv\ Y) \vee \neg Y) = 1.$

3 Computing diagnoses and conflicts

In order to define the concepts of a diagnosis and a conflict, we assume that an observation, a ground instance of *OBS*, is given. In the following definitions $\forall F$ denotes universal closure, i.e., all free variables in the formula F are universally quantified.

Definition. An *ok-instance* of a term is an instance where some variables are replaced by the constant *ok*. A *ground instance* is an instance where all the variables are replaced by constants.

Definition. A *diagnosis D* for $\langle SD,\ COMPS,\ OBS\rangle$ is an instance of *COMPS* such that $SD \models \forall m(D,\ OBS)$.

Definition. A *conflict C* for $\langle SD,\ COMPS,\ OBS\rangle$ is an *ok-instance* of *COMPS* such that $SD \models \forall\neg m(C,\ OBS)$.

This characterization of a diagnosis subsumes most of the previous ones. In the consistency-based approach (Reiter 1987) a diagnosis[2] is a set of abnormal ($\neq$*ok*) components such that *SD* and *OBS* are consistent with all other components being *ok*. De

[2]Reiter's definition of a diagnosis actually includes the minimality criterion and corresponds to our definition of a minimal diagnosis.

Kleer & Williams (1989) extended the definition to include a behavioral mode (state) for each component. In both cases a diagnosis is essentially a ground instance of *COMPS*. Poole (1989) observed that a diagnosis need not commit a state to each component when the state is 'don't care'. This led to the definition of a partial diagnosis (de Kleer *et al.* 1990) which corresponds to a non-ground instance of *COMPS* but, on the other hand, does not include states of components. The definition of a conflict is standard, i.e., a set of components which cannot be simultaneously *ok*, and can be easily extended to a minimal conflict.

Apart from being simple, our definitions are also operational since diagnoses and conflicts can easily be computed by a logic programming system. The search for a logical consequence of *SD* is realized by the search for an answer substitution Θ such that $SD \cup \{\neg m(A\Theta,\ OBS)\}$ is unsatisfiable. We just have to make sure that A is an ok-instance of *COMPS*. If such a substitution exists we can conclude $D = A\Theta$ is a diagnosis. If not, and regarding *SD* as implicitly completed (Lloyd 1987), we can conclude that $C = A$ is a conflict. Like in consistency-based diagnosis, A can be interpreted as an assumption that some components are not abnormal, i.e., are *ok*.

Example. Take the two inverters example, the boolean specification of the fault models, and the observation $OBS = \langle 1,0 \rangle$. In the case of the strong fault model we get the following diagnosis:

$\leftarrow\ m(\langle I1,I2\rangle,\ \langle 1,0\rangle).$
$I1 = W$
$I2 = 1$

This means that $I2$ is certainly faulty, while $I1$ might or might not be *ok*. With the exoneration model we get:

$\leftarrow\ m(\langle I1,I2\rangle,\ \langle 1,0\rangle).$
$I1 = Y$
$I2 = 1 \oplus Y$

This is equivalent to $I1 = \neg I2$, i.e., either $I1$ or $I2$ is faulty, but not both. The weak fault model yields:

$\leftarrow\ m(\langle I1,I2\rangle,\ \langle 1,0\rangle).$
$I1 = Y \oplus Y \wedge W1 \oplus W1$
$I2 = 1 \oplus Y \oplus Y \wedge W2$

which is equivalent to $I1 \vee I2 = 1$. The observation can be explained by each individual inverter being faulty, or by both inverters being faulty. In such cases one seeks a parsimonious diagnosis, i.e., a *minimal* number of faulty components which are still consistent with observations.

In order to compute minimal diagnoses it is convenient to represent the search space of diagnoses and conflicts as a subset-superset lattice (de Kleer & Williams 1987). The top element of the lattice corresponds to a tuple where all components are $\neq ok$, and the bottom element to the tuple where all components are *ok*. Reiter's algorithm (1987), for example, searches the lattice bottom-up (from conflicts to diagnoses), in a breadth-first fashion (diagnoses of smaller cardinality are found first) and relies on the underlying ATMS-like theorem prover. In contrast, our algorithm (Mozetic & Holzbaur 1991b) better fits into the logic programming environment, and implements a top-down, depth-first search through the lattice. In diagnosing real systems, the search lattice is large and minimal diagnoses are usually near the bottom of the lattice. Depth-first search, coupled with non-ground model calls, allows for deep 'dives' into the lattice and in the average case at least a few diagnoses are found quickly. Further, by computing minimal diagnoses incrementaly, we can ensure that the worst case complexity of the algorithm remains polynomial.

4 Boolean domains — using CLP($\mathcal{B}$)

In this section we show how a richer computational domain than the Herbrand universe interacts with a model and the diagnostic algorithm. We stay in the context of logic programming but extend syntactic unification by solving equations over interpreted terms (Jaffar *et al.* 1986) — boolean expressions in this particular case. The resulting constraint logic programming language CLP($\mathcal{B}$) was realized in the general Metaprolog framework (Holzbaur 1990), where the implementation reduces to the *Prolog* formulation of a specialized unification algorithm. There exist several boolean unification algorithms (Crone-Rawe 1989). We chose the one published by (Büttner & Simonis 1987); the origin of the method goes back to (Boole 1947). The algorithm computes the *most general boolean ring unifier* θ of two terms t_1 and t_2. It operates on a deterministic disjunctive minimal normal form (Martin & Nipkov 1986) for terms in the boolean ring $\langle V,\oplus,\wedge,0,1\rangle$, i.e., all boolean functions are expressed in terms of $\oplus$ and $\wedge$.

Example.
$\neg X \rightarrow 1 \oplus X$
$X \vee Y \rightarrow X \oplus Y \oplus X \wedge Y$

The use of CLP($\mathcal{B}$) is limited to discrete devices which compute some boolean function. However, it allows for a compact representation of the search lattice, since a minimal diagnosis can be computed deterministically, without any backtracking.

Example (binary adder, Genesereth 1984, Figure 3). The top-level binary predicate m is *adder*, *COMPS* is a five-tuple $\langle X1,X2,A1,A2,O1\rangle$, *OBS* is a five-tuple $\langle A,B,C,\ D,E\rangle$, and *SD* consists of the following logic program:

```
adder( ⟨X1,X2,A1,A2,O1⟩, ⟨A,B,C, D,E⟩)  ←
        xorg( X1, A, B, X ),
        xorg( X2, C, X, D ),
        andg( A1, A, B, Y ),
```

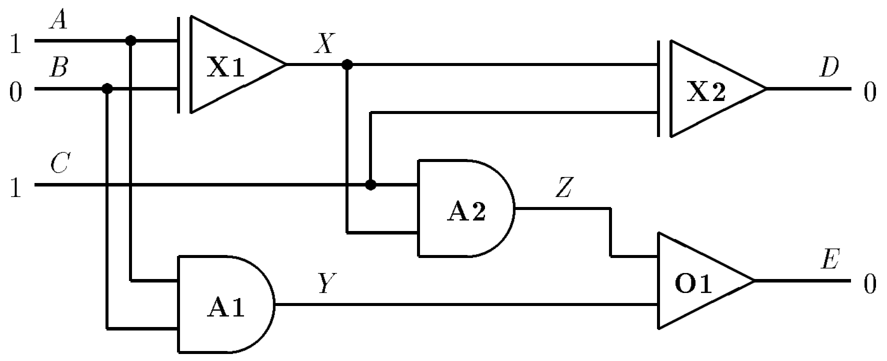

Figure 3: A binary adder consisting of two EXCLUSIVE-OR gates (X1, X2), two AND gates (A1, A2) and an OR gate (O1). The output $E = 0$ is faulty.

andg(A2, C, X, Z),
org(O1, Y, Z, E).

xorg($(X \oplus Y \oplus Z) \vee W$*, X, Y, Z).*
andg($((X \wedge Y) \oplus Z) \vee W$*, X, Y, Z).*
org($((X \vee Y) \oplus Z) \vee W$*, X, Y, Z).*

Given the observation $\langle 1,0,1,\ 0,0\rangle$ and the above weak fault model of the adder, CLP($\mathcal{B}$) returns the following answer substitutions which constitute a solved system of boolean equations in the normal form:

$\leftarrow$ *adder(* $\langle X1,X2,A1,A2,O1\rangle$, $\langle 1,0,1,\ 0,0\rangle$*).*
$X1 = 1 \oplus X \oplus X \wedge W1$
$X2 = 1 \oplus X \oplus X \wedge W2$
$A1 = Y \oplus Y \wedge W3 \oplus W3$
$A2 = X \oplus X \wedge W4 \oplus Z \oplus Z \wedge W4 \oplus W4$
$O1 = Y \oplus Y \wedge Z \oplus Y \wedge Z \wedge W5 \oplus Y \wedge W5 \oplus Z \oplus Z \wedge W5 \oplus W5$

Note that this answer substitution captures *all* diagnoses, i.e., the whole lattice above the boundary between the diagnoses and conflicts. A minimal diagnosis is computed by setting a maximum number of state variables to *ok* (0).

Example. Assume that we already succeeded setting $X1, X2, A1$ to 0, which is logically equivalent to the model call:

$\leftarrow$ *adder(* $\langle 0,0,0,A2,O1\rangle$, $\langle 1,0,1,\ 0,0\rangle$*).*
$A2 = 1 \oplus Z \oplus Z \wedge W4$
$O1 = Z \oplus Z \wedge W5 \oplus W5$

Next, $A2$ is set to 0, forcing Z to 1 and $O1$ to 1, thus yielding the first minimal singleton diagnosis $\langle 0,0,0,0,1\rangle$, i.e., the or gate $O1$ is faulty. An alternative diagnosis is computed from a disjoint label $\langle X1,X2,A1,A2,0\rangle$, which yields the second minimal diagnosis $\langle 0,0,0,1,0\rangle$. In both cases we know that the diagnoses are minimal — there is no need to even consider the label $\langle 0,0,0,0,0\rangle$.

After the discovery of the two single faults the label $\langle X1,X2,A1,0,0\rangle$ leads to the model call with the following answer substitutions:

```
← adder( ⟨X1,X2,A1,0,0⟩, ⟨1,0,1, 0,0⟩).
X1 = 1
X2 = 1
A1 = W3
```

This corresponds to the minimal diagnosis $\langle 1,1,0,0,0\rangle$ — no need to check $\langle 1,0,0,0,0\rangle$ or $\langle 0,1,0,0,0\rangle$. In addition, under the assumption that both $A2$ and $O1$ are *ok* we known that neither $X1$ nor $X2$ can possibly be *ok*. This yields the two conflicts $\langle X1,0,A1,0,0\rangle$ and $\langle 0,X2,A1,0,0\rangle$. Therefore, from three calls to the model we got the three minimal diagnoses and even the two conflicts, which in total covers the search lattice.

5 Real-valued domains — using CLP($\Re$)

There are domains where pure logic programs or ATMS-like systems have insufficient expressive power to reason about the system under consideration. In particular, modeling real-valued system parameters with tolerances requires some degree of numerical processing, and feedback loops in general cannot be resolved by local constraint propagation methods. Examples of such systems are analogue circuits, such as amplifiers or filters (Dague *et al.* 1990).

Here we illustrate the modeling of a simple amplifier (taken from Wakeling & McKeon 1989) by CLP($\Re$). The description of a transistor we use is from Heinze *et al.* (1987). We assume that any transistor or resistor in the circuit can be faulty.

Example (an amplifier, Wakeling & McKeon 1989, Figure 4). The top-level binary predicate m is *amplifier*, *COMPS* is an eleven-tuple $\langle Q1,Q2,Q3,R1,\ldots,R8\rangle$, *OBS* is a four-tuple $\langle Vcc,Vee,Vin,Vout\rangle$, and *SD* consists of the following logic program:

```
amplifier( ⟨Q1,Q2,Q3,R1,R2,R3,R4,R5,R6,R7,R8⟩, ⟨Vcc,Vee,Vin,Vout⟩)  ←
        transistor( npn, Q1, V1, V3, V2, Ib1, Ic1, Ie1 ),
        transistor( npn, Q2, V6, V4, V2, Ib2, Ic2, Ie2 ),
        transistor( pnp, Q3, V4, Vout, V5, Ib3, Ic3, Ie3 ),
        resistor( R1, 10000, Vin, V1, I1 ),
        resistor( R2, 10000, Vout, V1, I2 ),
        resistor( R3, 6800, Vcc, V3, Ic1 ),
        resistor( R4, 6800, Vcc, V4, I4 ),
        resistor( R5, 560, Vcc, V5, Ie3 ),
```

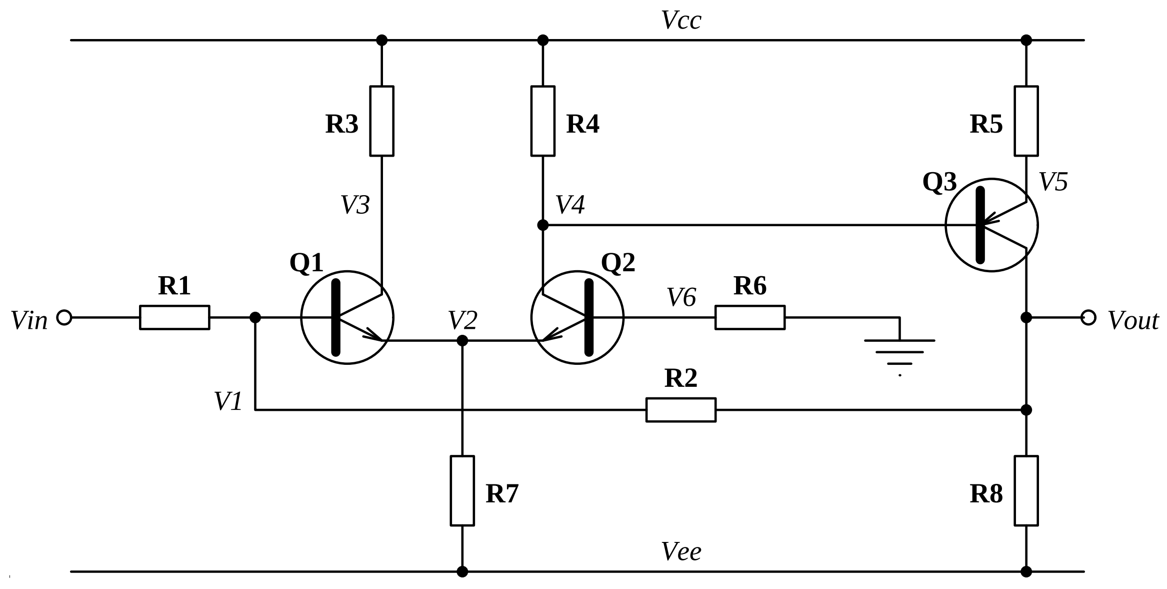

Figure 4: An amplifier consisting of transistors (Q1, Q2, Q3) and resistors (R1, …, R8).

```
        resistor( R6, 560, 0, V6, Ib2 ),
        resistor( R7, 15000, V2, Vee, I7 ),
        resistor( R8, 3300, Vout, Vee, I8 ),
        Ib1=I1+I2, I7=Ie1+Ie2, Ic2=I4+Ib3, Ic3=I2+I8.
resistor( ok, R, V1, V2, I ) ← V1−V2=I*R.
resistor( ab, _, _, _, _ ).
transistor( npn, cutoff, Vb, Vc, Ve, Ib, Ic, Ie ) ←
        Vb<Ve+0.7, Ib=0, Ic=0, Ie=0.
transistor( npn, active, Vb, Vc, Ve, Ib, Ic, Ie ) ←
        Vb=Ve+0.7, Vc≥Vb, Ib≥0, Ic=100*Ib, Ie=Ic+Ib.
transistor( npn, saturated, Vb, Vc, Ve, Ib, Ic, Ie ) ←
        Vb=Ve+0.7, Vc=Ve+0.3, Ib≥0, Ic≥0, Ie=Ic+Ib.
transistor( _, ab, _, _, _, _, _, _ ).
```

In CLP($\Re$) linear equations are kept in *solved form*. Variables appearing in the equations are split into two disjoint sets: *dependent* variables and *independent* variables. Dependent variables are expressed through terms containing independent variables. When a new equation is to be combined with a system of equations in solved form, all its dependent variables are replaced by their definitions which results in an expression over independent variables. An independent variable is selected then, and the expression is solved for it. After the resulting definition has been back-substituted into the equation system, the isolated variable can be added as a new dependent variable, and the equation system is in solved form again. Inequalities are expressed in terms of independent variables.

One traditional method for deciding linear inequalities is the simplex method. The simplex method works by turning inequalities into equations through the introduction of so-called 'slack variables'. This leads to a 'contamination' of the equation system with artificial variables from the user's point of view. In our experience, the amount of code that is needed to compute a human readable form of the (in)equation system is unproportionally high in comparison to the code that does the actual job. Therefore, we rather selected the Shostak's 'Loop Residue' method (Shostak 1981). Besides being better suited for small inequalities, this method operates with a 'direct' representation of inequalities. For each set of inequalities a graph is constructed whose vertices correspond to the variables and edges to the inequalities. Kraemer (1989) proves the equivalence between a satisfiable set of inequalities and the corresponding closed graph without any infeasible loop.

Our Metaprolog implementation of CLP($\Re$) is preferred over other existing implementations of CLP($\Re$) (Heintze *et al.* 1987, Jaffar 1990) since it allows for the simultaneous use of solvers for different domains in a consistent framework. In this respect Metaprolog is very well suited for the computational demands that arise in the context of hierarchical abstractions (Mozetic 1990, Mozetic & Holzbaur 1991a). The numerical level of the model can be formulated with CLP($\Re$) for example, and successive qualitative abstractions thereof typically utilize constraint propagation over finite domains where forward checking or CLP($\mathcal{B}$) can be used.

6 Conclusion

The paper outlines the potential applicability of Constraint Logic Programs to model-based diagnosis. For the CLP($\mathcal{B}$) case we show how the diagnostic algorithm can actively benefit from an increased expressiveness in the language. Another solver, CLP($\Re$), is able to solve systems of simultaneous linear equations over $\Re$eals. This is beyond the capabilities of the local constraint propagation methods used by de Kleer & Williams (1987). In our framework the simultaneous application of several specific solvers is truly operational due to their realization in the uniform Constraint Logic Programming scheme.

Acknowledgements

This work was supported by the Austrian Federal Ministry of Science and Research. Thanks to Robert Trappl for creating an enjoyable working environment.

References

Boole, G. (1947). *The Mathematical Analysis of Logic.* Macmillan.

Büttner, W., Simonis, H. (1987). Embedding Boolean expressions into logic programming. *Journal of Symbolic Computation 4*, pp. 191-205.

Cohen, J. (1990). Constraint logic programming languages. *Communications of the ACM 33 (7)*, pp. 52-68.

Crone-Rawe, B. (1989) Unification algorithms for boolean rings. SEKI Working Paper SWP-89-01, University of Kaiserslautern, Germany.

Dague, P., Deves, P., Luciani, P., Taillibert, P. (1990). Analog systems diagnosis. *Proc. 9th ECAI*, pp. 173-178, Stockholm.

Davis, R. (1984). Diagnostic reasoning based on structure and behaviour. *Artificial Intelligence 24*, pp. 347-410.

de Kleer, J., Mackworth, A.K., Reiter, R. (1990). Characterizing diagnoses. *Proc. 8th AAAI*, pp. 324-330, Boston, MIT Press.

de Kleer, J., Williams, B.C. (1987). Diagnosing multiple faults. *Artificial Intelligence 32*, pp. 97-130.

de Kleer, J., Williams, B.C. (1989). Diagnosis with behavioral modes. *Proc. 11th IJCAI*, pp. 1324-1330, Detroit, Morgan Kaufmann.

de Kleer, J., Williams, B.C. (1990). Focusing the diagnosis engine. Unpublished draft, presented at the *First Intl. Workshop on Principles of Diagnosis*, Stanford University.

Friedrich, G., Gottlob, G., Nejdl, W. (1990). Hypothesis classification, abductive diagnosis and therapy. *Proc. First Intl. Workshop on Principles of Diagnosis*, pp. 124-128, Stanford University.

Gallanti, M., Roncato, M., Stefanini, A., Tornielli, G. (1989). A diagnostic algorithm based on models at different level of abstraction. *Proc. 11th IJCAI*, pp. 1350-1355, Detroit, Morgan Kaufmann.

Genesereth, M.R. (1984). The use of design descriptions in automated diagnosis. *Artificial Intelligence 24*, pp. 411-436.

Heintze, N., Jaffar, J., Michaylov, S., Stuckey, P., Yap, R. (1987). The CLP($\Re$) programmer's manual. Dept. of Computer Science, Monash University, Australia.

Heintze, N., Michaylov, S., Stuckey, P. (1987). CLP($\Re$) and some electrical engineering problems. *Proc. 4th Intl. Conference on Logic Programming*, pp. 675-703, Melbourne, Australia, The MIT Press.

Holzbaur, C. (1990). Specification of constraint based inference mechanisms through extended unification. Ph.D. Thesis, Vienna University of Technology, Austria.

Jaffar, J. (1990). CLP($\Re$) version 1.0 reference manual. IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY.

Jaffar, J., Lassez, J.-L., Mahler, J. (1986). A logic programming language scheme. In D. de Groot, G. Linstrom (eds.), *Logic Programming: Functions, Relations, and Equations*, Prentice-Hall, Englewood Cliffs, NJ.

Kraemer, F.-J. (1989). A decision procedure for Presburger arithmetic with functions and equality. SEKI working paper SWP-89-4, FB Informatik, University of Kaiserslautern, Germany.

Lloyd, J.W. (1987). *Foundations of Logic Programming* (Second edition). Springer-Verlag, Berlin.

Martin, U., Nipkov, T. (1986). Unification in boolean rings. *Proc. 8th Intl. Conference on Automated Deduction*, pp. 506-513.

Mozetic, I. (1990). Reduction of diagnostic complexity through model abstractions. Report TR-90-10, Austrian Research Institute for Artificial Intelligence, Vienna. *Proc. First Intl. Workshop on Principles of Diagnosis*, pp. 102-111, Stanford University, Palo Alto.

Mozetic, I., Holzbaur, C. (1991a). Integrating qualitative and numerical models within Constraint Logic Programming. Report TR-91-2, Austrian Research Institute for Artificial Intelligence, Vienna, Austria. *Proc. 1st European Workshop on Qualitative Reasoning about Physical Systems*, Genova, Italy.

Mozetic, I., Holzbaur, C. (1991b). Controlling the complexity in model-based diagnosis. Report TR-91-3, Austrian Research Institute for Artificial Intelligence, Vienna, Austria.

Poole, D. (1989). Normality and faults in logic-based diagnosis. *Proc. 11th IJCAI*, pp. 1304-1310, Detroit, Morgan Kaufmann.

Raiman, O. (1989). Diagnosis as a trial: the alibi principle. IBM Scientific Center, Paris.

Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence 32*, pp. 57-95.

Shostak, R. (1981). Deciding linear inequalities by computing loop residues. *Journal of the ACM 28 (4)*, pp. 769-779.

Struss, P., Dressler, O., (1989). "Physical negation" — integrating fault models into the general diagnostic engine. *Proc. 11th IJCAI*, pp. 1318-1323, Detroit, Morgan Kaufmann.

Wakeling, A., McKeon A. (1989). On automatic fault finding in analogue circuits. *Electronic Engineering*, pp. 95-101, Nov. 1989.