Integrating Numerical and Qualitative Models within Constraint Logic Programming[1]

Igor Mozetič
Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A-1010 Vienna, Austria
igor@ai-vie.uucp

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
christian@ai-vie.uucp

Abstract

The paper describes an interplay between numerical and qualitative models represented in a uniform Constraint Logic Programming framework. In the context of model-based diagnosis a detailed, numerical model is used to discriminate between competing diagnoses at the abstract, qualitative level. A distinguishing feature of our approach is that the abstract proof is used to guide the verification at the detailed level, and to refute impossible refinements as soon as possible by introducing additional constraints on the variables. An implemented instance of this framework, CLP($\Re$), which is used in the paper, comprises a solver for systems of linear equations and inequalities over real-valued variables.

1 Introduction

In model-based approach to diagnosis one starts with a model of a real-world system which explicitly represents the structure and behavior of components of the system (e.g., de Kleer & Williams 1987, Reiter 1987). When the system's actual behavior differs from the expected behavior, the diagnostic problem arises. The model is then used to identify the set of faulty components which account for the observed behavior.

In our view there are two major obstacles which prevented a wider application of model-based techniques to real-world problems. First, models are usually restricted to qualitative descriptions. GDE (de Kleer & Williams

[1]Appears in *Proc. 1991 Intl. Logic Programming Symposium, ILPS-91* (V. Saraswat, K. Ueda, Eds.), pp. 678-693, San Diego, CA, October 28-31, 1991, The MIT Press.

1987), for example, is unable to solve simultaneous equations, which makes it unpractical for a large class of applications. Second, 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 1991a), by interleaving diagnosis and treatment (Friedrich *et al.* 1990), or by introducing abstractions (Gallanti *et al.* 1989, Mozetic 1990).

In the paper we address both issues, the integration of numerical and qualitative models, and the reduction of diagnostic complexity through abstractions. The following is an outline of our approach:

1. specification of a numerical model of the normal behavior of the system,
2. specification of abstraction operators,
3. automatic derivation of an abstract, qualitative model,
4. addition of a weak fault model (which makes no specific assumption about the abnormal behavior) to the normal behavior model,
5. for given observations, computing minimal diagnoses at the abstract level,
6. discriminating between alternative abstract diagnoses by referring back to the numerical model,
7. using the abstract proof to refute impossible refinements.

The emphasis of the paper is on items 6 and 7, i.e., on the interplay between the qualitative and numerical models. While the whole scheme is domain independent, we illustrate it on an example of an electrical circuit — an OR gate realized with three transistors.

In section 2 we present CLP($\Re$), a language suitable for numerical modeling within the logic programming framework (Holzbaur 1990). It should be noted that CLP($\Re$) is only an instance of the general Constraint Logic Programming scheme (Jaffar *et al.* 1986).

In section 3 we define three abstraction operators. They specify how a detailed, numerical model can be simplified by collapsing constants (thus defining a quantity space), by deleting irrelevant arguments (i.e., model parameters), and by unfolding non-operational predicates (i.e., ignoring the internal structure of some model components). Abstraction operators are then used to automatically derive an abstract, qualitative model of the system (Mozetic & Holzbaur 1991b). Abstractions turned out to be useful in reducing the search space in theorem proving (Plaisted 1981, Giunchiglia & Walsh 1989), planning (Sacerdoti 1974, Korf 1987), and in model-based diagnosis (Gallanti *et al.* 1989, Mozetic 1990).

Section 4 outlines the hierarchical diagnosis. Since the original, numerical model and the derived, qualitative model comprise just the normal behavior of the system, first a weak fault model is added. This makes no assumption about how components can fail, and is characteristic for the consistency-based approach to diagnosis (e.g., Reiter 1987, de Kleer & Williams 1987). For given observations, different than expected, the diagnostic algorithm refers to the qualitative model and computes abstract diagnoses. These identify minimal sets of components which, if assumed to be faulty, render the model behavior consistent with the observations. However, there might be alternative, competing diagnoses, and the abstract model cannot discriminate between them without additional information. At this point, one can refer back to the detailed, numerical model.

Section 5 addresses the mapping back of the abstract proof (Giunchiglia & Walsh 1990) and constraint resolution at the numerical level. Not only are the abstract diagnoses refined by applying the abstraction operators in the 'inverse' direction, but the abstract proof is used as well. Its role is to refute impossible derivations at the detailed level as soon as possible. A refutation amounts to determining the unsatisfiability of a system of linear equations and inequalities over real-valued variables, and is realized by the Shostak's 'Loop Residue' algorithm (Shostak 1981).

2 Modeling with CLP($\Re$)

We represent models by Constraint Logic Programs (CLP, Jaffar *et al.* 1986) which are logic programs extended by interpreted functions. A proper implementation of a CLP scheme allows for an easy integration of specialized problem solvers into the logic programming framework. For example, in our implementation (Holzbaur 1990) specialized solvers communicate with the standard Prolog interpreter via extended semantic unification and are implemented in Prolog themselves. So far, three solvers have been implemented: 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 $\Re$eals.

We illustrate the diagnostic algorithm on a model of an OR gate. The initial numerical model of an OR gate (predicate org_n) is first abstracted into a qualitative model (org_q, Figure 1), and then both are used for diagnosis. The OR gate is realized by three npn transistors and resistors. The model relates qualitative states of transistors to real-valued voltages and currents. The model components are represented by atomic formulas, and connections between the components by shared variables.

The description of an npn transistor is from Heintze, Michaylov & Stuckey (1987b). The transistor operates in three states: *cutoff*, *saturated*, and *active*. In digital circuits usually only the cutoff and saturated states are of interest, and therefore the active state, relevant in amplifier circuits, is omit-

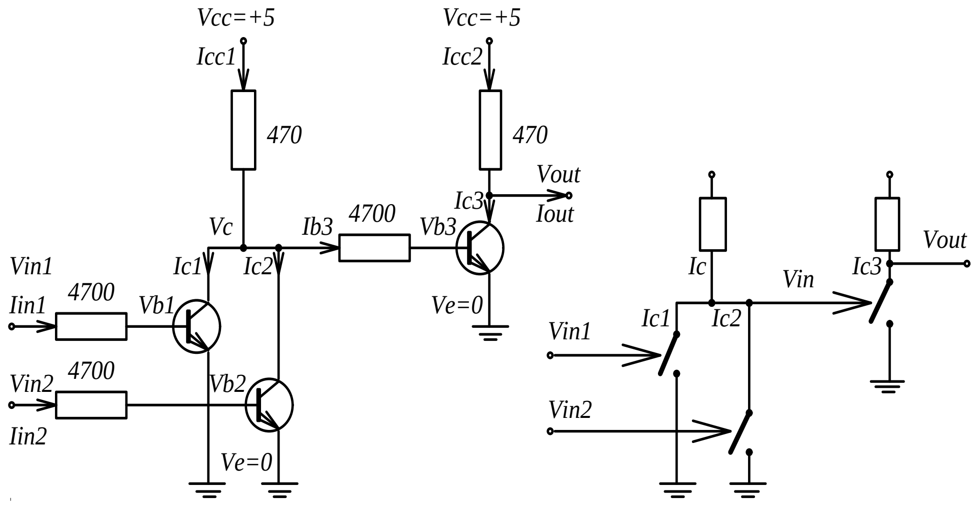


Figure 1: A numerical and a qualitative model of an OR gate.

ted. *Vx* and *Ix* denote the real-valued voltages and currents for the base, collector and emmiter, respectively. Constants *Beta*, *Vbe*, and *Vcesat* are device parameters.

```
$org_n$( comps(S1,S2,S3), obs(X,Y,Z) )  $\leftarrow$
        $norg_n$( S1, S2, X, Y, W ),
        $inv_n$( S3, W, Z ).

$norg_n$( S1, S2, b(Vin1,Iin1), b(Vin2,Iin2), b(V,I) )  $\leftarrow$
        $switch_n$( S1, Vin1, Iin1, V, Ic1 ),
        $switch_n$( S2, Vin2, Iin2, V, Ic2 ),
        $Ic1+Ic2=Ic$,
        $power_n$( Ic, V, I ).

$inv_n$( S, b(V,I), b(Vout,Iout) )  $\leftarrow$
        $switch_n$( S, V, I, Vout, Ic ),
        $power_n$( Ic, Vout, Iout ).

$switch_n$( S, Vin, Iin, Vc, Ic )  $\leftarrow$
        $Ve=0$, $Beta=100$, $Vbe=0.7$, $Vcesat=0.3$,
        $resistor_n$( Vin, Vb, Iin, 4700 ),
        $transistor_n$( S, Beta, Vbe, Vcesat, Vb, Vc, Ve, Iin, Ic, Ie ).

$power_n$( Ic, Vout, Iout )  $\leftarrow$
        $Vcc=5$, $Ic+Iout=Icc$,
        $resistor_n$( Vcc, Vout, Icc, 470 ),
        $0 \leq Iout$, $Iout \leq 0.006$.

$resistor_n$( V1, V2, I, R )  $\leftarrow$  $R>0$, $V1-V2=I*R$.

$transistor_n$( cutoff, Beta, Vbe, Vcesat, Vb, Vc, Ve, Ib, Ic, Ie )  $\leftarrow$
        $Vb<Ve+Vbe$, $Ib=0$, $Ic=0$, $Ie=0$.
```

$$\textit{transistor}_n(\textit{ saturated, Beta, Vbe, Vcesat, Vb, Vc, Ve, Ib, Ic, Ie }) \leftarrow$$
$$\textit{Vb=Ve+Vbe, Vc=Ve+Vcesat, } Ib \geq 0, \; Ic \geq 0, \; \textit{Ie=Ic+Ib.}$$

This model is unnecessarily detailed for a number of tasks. For diagnosis, for example, it does not really matter if a voltage is 4.4 or 4.6, but whether it is qualitatively *high* or *low*, and whether a transistor properly operates as a switching device. We can specify some currents as irrelevant, voltages 0–0.7 and 2–5 as indistinguishable, and derive a qualitative model of the OR gate.

In the following section we specify abstraction operators more precisely. We illustrate their applicability by providing an automatic abstraction of a numerical model of the OR gate into a qualitative model.

3 Abstraction operators

Underlying the formulation of abstractions is a typed logic program (Lloyd 1987). Types provide a natural way of expressing the concept of a domain and are convenient for specifying abstraction operators in a compact form. We assume that variables and constants have types such as τ. Functions have types of the form $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$, and predicates have types of the form $\tau_1 \times \ldots \times \tau_n$.

The following three abstraction operators capture all the possible ways in which atomic formulas can be simplified. They define a class of *atomic abstractions* and it turns out that these cover most of the previous work on abstractions in Artificial Intelligence (Giunchiglia & Walsh 1989). We use a binary predicate h_τ to denote abstractions of constants and functions of range type τ, and a binary predicate h to denote predicate abstractions.

1. Collapsing constants.
   Different constants can be renamed into a single constant. For example, assume that a_1 and a_2 are of type τ, and that they are collapsed into a single constant a':

   $h_\tau(a_1, a')$. $\qquad h_\tau(a_2, a')$.

2. Function abstractions.
   Functions can be renamed and irrelevant arguments deleted. For example, let f be of type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$, its first argument be deleted, and f be renamed to f':

   $h_\tau(f(X_1, X_2 \ldots, X_n), f'(X'_2, \ldots, X'_n)) \leftarrow h_{\tau 2}(X_2, X'_2), \ldots, h_{\tau n}(X_n, X'_n)$.

3. Predicate abstractions.
   Predicates can be renamed and some arguments deleted. For example, let p be of type $\tau_1 \times \ldots \times \tau_n$, its first argument be deleted, and p be renamed to p':

   $h(p(X_1, X_2 \ldots, X_n), p'(X'_2, \ldots, X'_n)) \leftarrow h_{\tau 2}(X_2, X'_2), \ldots, h_{\tau n}(X_n, X'_n)$.

The following specifies the three abstraction operators for the OR gate example.

Collapsing constants:

h_s*(cutoff, ok).*
h_s*(saturated, ok).*

h_i*(0, zero).*
h_i*(I, pos)* $\leftarrow$ *I>0.*

h_v*(V, low)* $\leftarrow$ *0≤V, V<0.7.*
h_v*(V, high)* $\leftarrow$ *2≤V, V≤5.*

Function abstractions:

h_c*(comps(S1,S2,S3), comps(S1',S2',S3'))* $\leftarrow$
    h_s*(S1,S1'),* h_s*(S2,S2'),* h_s*(S3,S3').*

h_o*(obs(X,Y,Z), obs(X',Y',Z'))* $\leftarrow$ h_b*(X,X'),* h_b*(Y,Y'),* h_b*(Z,Z').*

h_b*(b(V,_I), V')* $\leftarrow$ h_v*(V,V').*   *% I is deleted*

Predicate abstractions:

h(org$_n$*(Comps,Obs), org*$_q$*(Comps',Obs'))* $\leftarrow$
    h_c*(Comps,Comps'),* h_o*(Obs,Obs').*

h(norg$_n$*(S1,S2,X,Y,Z), norg*$_q$*(S1',S2',X',Y',Z'))* $\leftarrow$
    h_s*(S1,S1'),* h_s*(S2,S2'),* h_b*(X,X'),* h_b*(Y,Y'),* h_b*(Z,Z').*

h(inv$_n$*(S,X,Y), inv*$_q$*(S',X',Y'))* $\leftarrow$
    h_s*(S,S'),* h_b*(X,X'),* h_b*(Y,Y').*

h(switch$_n$*(S,Vin,_Iin,Vc,Ic), switch*$_q$*(S',Vin',Vc',Ic'))* $\leftarrow$
    h_s*(S,S'),* h_v*(Vin,Vin'),* h_v*(Vc,Vc'),* h_i*(Ic,Ic').*

h(power$_n$*(Ic,Vout,_Iout), power*$_q$*(Ic',Vout'))* $\leftarrow$
    h_i*(Ic,Ic'),* h_v*(Vout,Vout').*

h(X+Y=Z, sum$_q$*(X',Y',Z'))* $\leftarrow$
    h_i*(X,X'),* h_i*(Y,Y'),* h_i*(Z,Z').*

From the numerical model of the OR gate and the above abstractions, a qualitative model was automatically derived through term rewriting and partial evaluation. Predicates, for which no abstraction is specified (*resistor*$_n$, *transistor*$_n$) are treated as non-operational and their definitions are unfolded. The remaining predicates and terms are rewritten according to the abstraction operators. The derivation can be regarded as an enhancement of explanation-based generalization without a learning example (Van Harmelen & Bundy 1988). The relation to the explanation-based generalization and a detailed description of the abstraction algorithm is in Mozetic & Holzbaur (1991b). Here is the result of its application:

```
$org_q$( comps(S1,S2,S3), obs(Vin1,Vin2,Vout) ) ←
        $norg_q$( S1, S2, Vin1, Vin2, V ),
        $inv_q$( S3, V, Vout ).


$norg_q$( S1, S2, Vin1, Vin2, V ) ←
        $switch_q$( S1, Vin1, V, Ic1 ),
        $switch_q$( S2, Vin2, V, Ic2 ),
        $sum_q$( Ic1, Ic2, Ic ),
        $power_q$( Ic, V ).

$inv_q$( S, V, Vout ) ←
        $switch_q$( S, V, Vout, Ic ),
        $power_q$( Ic, Vout ).

$switch_q$( ok, low, _, zero ).
$switch_q$( ok, high, low, zero ).
$switch_q$( ok, high, low, pos ).

$power_q$( zero, high ).
$power_q$( pos, low ).
$power_q$( pos, high ).

$sum_q$( zero, zero, zero ).
$sum_q$( zero, pos, pos ).
$sum_q$( pos, zero, pos ).
$sum_q$( pos, pos, pos ).
```

4 Hierarchical diagnosis

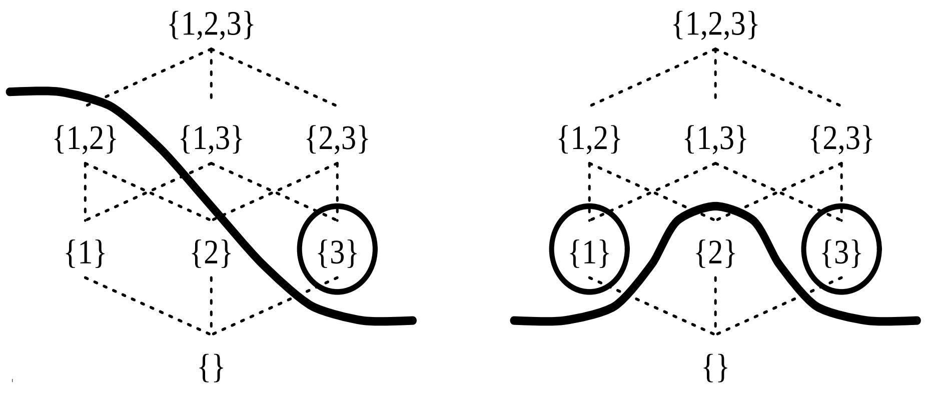
Both, the original and the abstract OR gate model are just models of *normal* behavior, and they do not specify anything about the possible failures. If we want to solve diagnostic tasks then a *fault model*, which specifies how a component behaves when it is faulty, has to be added. However, in many domains this is not known and one has to resort to a *weak* fault model. A weak fault model allows for *any* behavior and does not impose any constraints on faulty components.

In our case we assume that resistors cannot fail, and that all the components are correctly connected. Components which can fail are $transistor_n$ at the numerical level, and $switch_q$ at the qualitative level. For these two components, a weak fault model is defined by the following two clauses:

```
$transistor_n$( ab, _, _, _, _, _, _, _, _, _ ).

$switch_q$( ab, _, _, _ ).
```

Now assume that the voltages *Vin1* = 4.5 and *Vin2* = 0 were applied to the input of an actual OR gate, and the voltage *Vout* = 0.2 was measured at the output. Clearly, the OR gate does not perform the intended boolean



{1} ⋯ *comps(ab, ok, ok)*
{3} ⋯ *comps(ok, ok, ab)*

Figure 2: Search lattices and minimal diagnoses for the numerical (left) and the qualitative (right) model of the OR gate.

function (assuming positive logic), and we want to locate the faulty transistors. Note that in general there might be multiple faults. The augmented numerical model can be used for diagnosis directly, by submitting the following query to the CLP($\Re$) interpreter:

$\leftarrow$ *org_n($Diag_n$, obs(b(4.5,Iin1), b(0,Iin2), b(0.2,Iout))).*

However, in order to reduce the search space and to minimize a relatively expensive numerical computation, one can first use the abstract, qualitative model to find potential diagnoses. First, the observation is abstracted and the following query is submitted to the standard Prolog system:

$\leftarrow$ *org_q($Diag_q$, obs(high, low, low)).*

There are six answer substitutions (diagnoses) which satisfy the query. One of them, *$Diag_q$ = comps(ab,ab,ab)*, for example, states that all three transistors are faulty. Such answers are not very useful, since one wants just diagnoses which refer to a *minimal* number of faulty components. In order to find minimal diagnoses, an algorithm on top of the model interpreter was implemented (Mozetic & Holzbaur 1991a). The algorithm searches through the subset-superset lattice, where nodes correspond to different sets of components being faulty (Figure 2). The root {1,2,3} corresponds to all components being faulty, and the bottom node {} to all components being normal.

In the case of the qualitative OR gate model, the algorithm returns the following two *minimal* diagnoses:

$Diag_q$1 = comps(ab,ok,ok)
$Diag_q$2 = comps(ok,ok,ab)

The interpretation is that either the first or the third transistor is faulty.

$$\begin{tikzcd}[column sep=small]
 & & \textit{ab,ok,ok} \arrow[dll] \arrow[dl] \arrow[dr] \arrow[drr] & & \\
\textit{ab,cutoff,cutoff} & \textit{ab,cutoff,saturated} & & \textit{ab,saturated,cutoff} & \textit{ab,saturated,saturated}
\end{tikzcd}$$

Figure 3: The refinements of an abstract diagnosis resulting from the 'inverse' application of the h_c abstraction operator.

In order to discriminate between these two alternatives, the hierarchical diagnostic algorithm refers back to the original, numerical model.

There are four refinements of the first diagnosis (Figure 3):

$\mathit{Diag}_n\mathit{11}$ = *comps(ab,cutoff,cutoff)*
$\mathit{Diag}_n\mathit{12}$ = *comps(ab,cutoff,saturated)*
$\mathit{Diag}_n\mathit{13}$ = *comps(ab,saturated,cutoff)*
$\mathit{Diag}_n\mathit{14}$ = *comps(ab,saturated,saturated)*

but none of them is possible with respect to the numerical model. On the other hand, the second diagnosis has one refinement:

$\mathit{Diag}_n\mathit{23}$ = *comps(saturated,cutoff,ab)*

which is actually admitted by the numerical model.

The role of abstractions in diagnosis is twofold. First, the search space is reduced since only refinements of the abstract diagnoses have to be verified at the detailed level. For example, since *comps(ok,ab,ok)* is not a diagnosis at the abstract level, there is no need to consider its four refinements. With appropriate abstractions this may result in an exponential improvement of diagnostic efficiency, as reported by Mozetic (1990).

Second, when there is an abstract diagnosis, we can 'unabstract' its derivation and thus provide a frame within which all detailed level derivations should be confined. As a consequence, a detailed derivation which falls out of the designated frame can be immediately terminated and recognized as unsuccessful. In the next section we show how a proof of the first abstract diagnosis

$\mathit{Diag}_q\mathit{1}$ = *comps(ab,ok,ok)*

is used in proving that the first refinement

$\mathit{Diag}_n\mathit{11}$ = *comps(ab,cutoff,cutoff)*

is impossible at the detailed level.

5 Mapping back and constraint resolution with CLP($\Re$)

A distinguishing feature of our approach is that not only are the abstract diagnoses refined, but also the abstract *proof* is used to guide the proof or refutation at the detailed level. The idea of constructing abstractions and mapping back the proof was originally proposed by Plaised (1981) in the context of resolution systems, and recently extended by Giunchiglia & Walsh (1990) to any axiomatic formal system. We restrict ourselves to Constraint Logic Programs and SLD resolution (Lloyd 1987), and place special attention to the treatment of numerical constraints which arise during mapping back.

In the following we show how the first of the four possible refinements of $Diag_q 1$

$Diag_n 11 = comps(ab,cutoff,cutoff)$

is refuted by the numerical model and mapping back of the abstract proof (Figure 4).

Given the observation and the constraints from mapping back the constants from the abstract proof, the detailed proof for $norg_n$ succeeds with some constraints. We provide a more detailed description of the refutation of inv_n. The refinement of the abstract proof step

$inv_q(\ ok,\ high,\ low\ )$

leads to the following goal at the numerical level:

$inv_n(\ cutoff,\ b(V,I),\ b(0.2,Iout)\ )$

with the additional constraints

$C1)\ I = -0.00212766*V+0.0106383$
$C2)\ V \geq 2.18$
$C3)\ V \leq 5$

The constant *cutoff* in the goal is the refinement of *ok* at the qualitative level, and 0.2 comes from the observation. The additional constraints on I and V result from mapping back the abstract proof and the execution of the numerical model up to this point. Note that *C1* is equivalent to $V = 5-470*I$, a more 'natural' version of the equation from the user's point of view, but CLP($\Re$) does not have such preferences.

An inverter consists (is defined in terms) of a switch and a power supply at both, the detailed and the abstract level. Mapping back the abstract proof step

$switch_q(\ ok,\ high,\ low,\ pos\ )$

into

$switch_n(\ cutoff,\ V,\ I,\ 0.2,\ Ic\ )$

adds a constraint on Ic

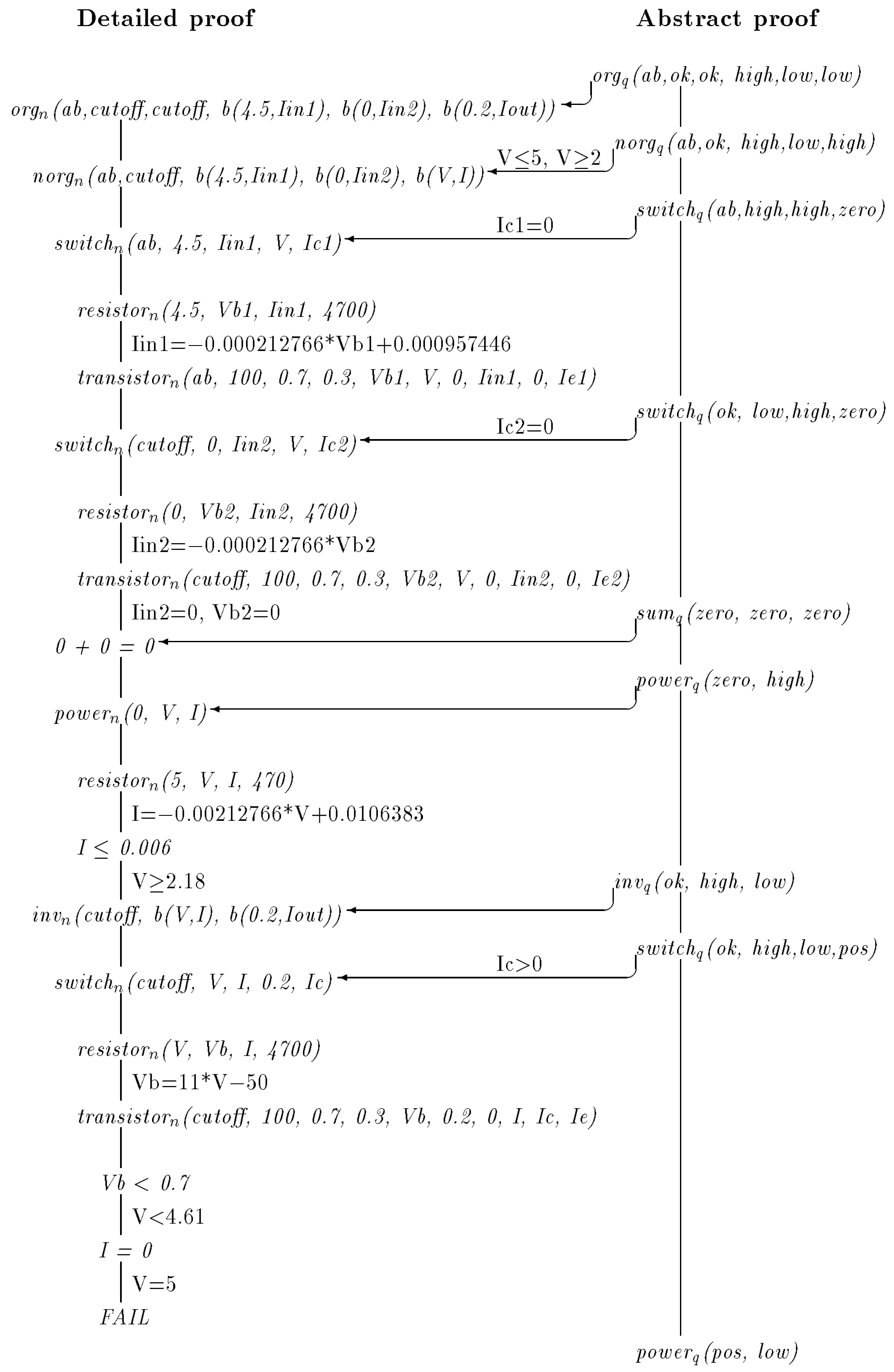

Detailed proof
Abstract proof
org_q(ab,ok,ok, high,low,low)
org_n(ab,cutoff,cutoff, b(4.5,Iin1), b(0,Iin2), b(0.2,Iout))
norg_q(ab,ok, high,low,high)
V≤5, V≥2
norg_n(ab,cutoff, b(4.5,Iin1), b(0,Iin2), b(V,I))
switch_q(ab,high,high,zero)
Ic1=0
switch_n(ab, 4.5, Iin1, V, Ic1)
resistor_n(4.5, Vb1, Iin1, 4700)
Iin1=−0.000212766*Vb1+0.000957446
transistor_n(ab, 100, 0.7, 0.3, Vb1, V, 0, Iin1, 0, Ie1)
switch_q(ok, low,high,zero)
Ic2=0
switch_n(cutoff, 0, Iin2, V, Ic2)
resistor_n(0, Vb2, Iin2, 4700)
Iin2=−0.000212766*Vb2
transistor_n(cutoff, 100, 0.7, 0.3, Vb2, V, 0, Iin2, 0, Ie2)
Iin2=0, Vb2=0
sum_q(zero, zero, zero)
0 + 0 = 0
power_q(zero, high)
power_n(0, V, I)
resistor_n(5, V, I, 470)
I=−0.00212766*V+0.0106383
I ≤ 0.006
V≥2.18
inv_q(ok, high, low)
inv_n(cutoff, b(V,I), b(0.2,Iout))
switch_q(ok, high,low,pos)
Ic>0
switch_n(cutoff, V, I, 0.2, Ic)
resistor_n(V, Vb, I, 4700)
Vb=11*V−50
transistor_n(cutoff, 100, 0.7, 0.3, Vb, 0.2, 0, I, Ic, Ie)
Vb < 0.7
V<4.61
I = 0
V=5
FAIL
power_q(pos, low)

Figure 4: Mapping back and accumulation of constraints.

C4) $Ic > 0$

A switch at the numerical level consists of a resistor and a transistor. When the detailed level goal

$transistor_n$(cutoff, 100, 0.7, 0.3, Vb, 0.2, 0, I, Ic, Ie)

is to be selected for resolution, the following constraints have been collected so far:

C1) $I = -0.00212766*V + 0.0106383$
C2) $V \geq 2.18$
C3) $V \leq 5$
C4) $Ic > 0$
C5) $Vb = 11*V - 50$

Some constants in the goal are from the definition of *$switch_n$* and the constraint *C5* was added by *$resistor_n$*. Before we proceed with the refutation, we sketch the operation of our CLP($\Re$) implementation (Holzbaur 1990).

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.

In our particular system of equations *I* and *Vb* are the only dependent variables. All the remaining variables are independent. Now assume that the head of the following clause is unified with the goal above:

$transistor_n$(cutoff, Beta, Vbe, Vcesat, Vb, Vc, Ve, Ib, Ic, Ie) $\leftarrow$
  G1) $Vb < Ve + Vbe$,
  G2) $Ib = 0$,
  G3) $Ic = 0$, $Ie = 0$.

The execution of the program proceeds with the subgoals of the *$transistor_n$* clause. The first subgoal is an inequality constraint. Our implementation of CLP($\Re$) decides the satisfiability of a system of linear inequalities by a version of the Shostak's 'Loop Residue' algorithm (Shostak 1981, Kraemer 1989). Each inequality is represented as an edge in the inequality graph $\mathcal{G}$. The algorithm only deals with loops in $\mathcal{G}$. For each loop, the residual inequality of the loop is computed and entered as a new edge into $\mathcal{G}$. The loop residue computation is iterated until no more loops can be created, or one of these new edges is determined to correspond to an unsatisfiable inequality. Each loop residue computation essentially eliminates one variable — therefore an unsatisfiable inequality will eventually result in a ground inequality $k < 0$, where k is a positive constant. The basic algorithm was extended to strict and nonstrict inequalities.

In our example, a graph $\mathcal{G}$ encodes the inequalities that arise during the evaluation. Initially there are just two edges corresponding to $V \leq 5$ and $V \geq 2.18$.

The first goal *(G1)* from the body of $transistor_n$ adds a new edge to $\mathcal{G}$. Inequalities of the form $A < B$ are normalized into $A - B < 0$. If the terms A and B contain dependent variables, their definitions are used. In our case *(G1)*

$$Vb < Ve + Vbe$$

is reexpressed in terms of V and values of Ve and Vbe

$$11*V - 50 < 0 + 0.7$$

which is normalized into

$$V - 4.61 < 0$$

This new edge of $\mathcal{G}$ subsumes the old inequality $V \leq 5$. The loop residue algorithm has been extended to narrow interval boundaries. Therefore the stronger bound $V < 4.61$ replaces the old one — edge $V \leq 5$ is essentially deleted from $\mathcal{G}$.

The subgoal selected next *(G2)* is $Ib = 0$. As Ib is unified to I, a dependent variable, the *C1* constraint reads as:

$$0 = -0.00212766*V + 0.0106383$$

which instantiates V to 5. This creates an infeasible loop in $\mathcal{G}$ since $5-4.61 = 0.39$ and $0.39 < 0$, which is clearly unsatisfiable. Therefore the original goal $transistor_n$ from the above and consequently $switch_n$ and inv_n fail. Without the constraints from the abstract proof, the refutation of inv_n for the given observation would have taken additional proof steps.

In general, the utilization of an abstract proof increases the number of constraints that have to be processed at the detailed level — by CLP($\Re$) in our example. Although tighter constraints correspond to a reduction of search space, the additional constraints have to be processed after all. The ultimate question whether one can benefit from hierarchical diagosis depends heavily on the application which determines the algebraic domains for the models. This in turn leads to the selection of appropriate decision procedures for the algebras. The computational complexity of these decision algorithms as a function of both the number and 'size' of equations or general constraints is the key to the answer of this question.

In our concrete example the utilization of abstractions leads to simpler constraints. The metrics measures the simplicity of constraints in terms of the number of variables they involve. Although our CLP($\Re$) solver has to deal with more constraints, they get simpler when the restrictions from the abstract level are added. One of the optimization methods that can be applied in an implementation of CLP($\Re$) is to have specialized code for constraints over 0,1,2,... variables. Therefore it is reasonable to expect better performance on smaller constraints. If the CLP($\Re$) solver is a logic program as in our implementation, this specializations can be produced quickly, safely and elegantly by partial evaluation.

In total, the role of CLP($\Re$) in the context of hierarchical diagnosis is manyfold. First, it allows to compute with a domain theory over real-valued variables at the numerical level of the model. Second, it admits the specification of abstraction operators that share the computational domain with the numerical model. The third function is to guarantee the satisfiablity of the numerical constraints collected during abstraction and mapping back.

Our implementation of CLP($\Re$) is preferred over existing versions (Heintze *et al.* 1987a, Jaffar 1990) since it allows for the simultaneous use of solvers for different domains in a consistent framework. This suits well the computational demands that arise in the context of hierarchical abstractions. The numerical level of the model can be formulated with CLP($\Re$) for example, and successive abstractions thereof typically utilize constraint propagation over finite domains. The implementation of the specialized solvers is based on user-definable extended unification. As the solvers are written in Prolog, they can easily be customized to specific demands. The choice of Prolog as an implementation language for the equation solver for CLP($\Re$) led to a reduction in code size by an order of magnitude.

Beside the principal (software engineering) issues that motivated our implementation of CLP($\Re$), the availability and the quality of Sicstus Prolog (Carlsson & Widen 1990) somehow aposteriori justified the selection of Prolog as an implementation language. Sicstus Prolog has a compiler which can produce native machine code and a garbage collector. The basic mechanisms provided for the implementation of *freeze/2* and *dif/2* are very useful for the implementation of extended unification, the basis of our approach.

Our first CLP($\Re$) implementation was based on the C-Prolog interpreter (Holzbaur 1990). For the performance comparison against the C implementations of CLP($\Re$) this was disadvantageous, as the unification extensions, i.e., the CLP($\Re$) solver, were interpreted only. However, given the Sicstus compiler, the performance of our current Prolog CLP($\Re$) implementation is somewhere in-between the IBM (Jaffar 1990) and the Monash (Heintze *et al.* 1987a) implementations. A further improvement of our version of CLP($\Re$), which did not require any extra effort from our side, accrues from the increased numerical precision in floating point operations in Sicstus (double precision). Since Sicstus also provides infinite precision integer arithmetics, the implementation of CLP($\mathcal{Q}$) ($\mathcal{Q}$ = rationals) is easy and reasonably efficient.

6 Conclusion

The contribution of this paper is a relatively self-contained presentation of the integration of qualitative and numerical models in the context of hierarchical diagnosis. Since hierarchical models, abstraction operators, the abstraction algorithm, and the diagnostic algorithm are expressed as (constraint) logic programs, we are able to provide a very concise and compact specification of these components. This in turn allows for the exposition of the interplay between the different abstraction levels in some depth.

We cover the specification of a numerical model, the specification of abstraction operators, hierarchical diagnosis, and in particular detail the 'flow' and the resolution of constraints between the numerical and the qualitative model. Because of the logically sound treatment of the abstraction operators by the underlying computational domain, we are able to use them in any direction. In this paper we concentrate on the 'mapping back' of constraints from the qualitative to the numerical level.

Acknowledgements

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

References

Carlsson, M., Widen, J. (1990). Sicstus Prolog user's manual, SICS/R-88/88007C, Swedish Institute of Computer Science, Kista, Sweden.

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

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.

Giunchiglia, F., Walsh, T. (1989). Abstract theorem proving. *Proc. 11th IJCAI*, pp. 372-377, Detroit, Morgan Kaufmann.

Giunchiglia, F., Walsh, T. (1990). Abstract theorem proving: mapping back. IRST Technical Report 8911-16, Trento, Italy.

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

Heintze, N., Michaylov, S., Stuckey, P. (1987b). 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, Technical University of Vienna, 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.

Korf, R.E. (1987). Planning as search: a quantitative approach. *Artificial Intelligence 33*, pp. 65-88.

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

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

Mozetic, I. (1990). Reduction of diagnostic complexity through model abstractions. *Proc. First Intl. Workshop on Principles of Diagnosis*, pp. 102-111, Stanford University.

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

Mozetic, I., Holzbaur, C. (1991b). Extending explanation-based generalization by abstraction operators. In *Machine Learning — EWSL-91* (Y. Kodratoff, Ed.), pp. 282-297, Porto, Portugal, Springer-Verlag (LNAI 482).

Plaisted, D.A. (1981). Theorem proving with abstractions. *Artificial Intelligence 16*, pp. 47-108.

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

Sacerdoti, E.D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence 5*, pp. 115-135.

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

Van Harmelen, F., Bundy, A. (1988). Explanation-based generalisation = partial evaluation. *Artificial Intelligence 36*, pp. 401-412.