Interval arithmetic with CLP($\Re$)

Igor Mozetič & Christian Holzbaur
Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A-1010 Vienna
Austria
(igor, christian)@ai.univie.ac.at

Abstract

We describe two extensions of CLP($\Re$), motivated by an application to model-based diagnosis of active analog filters. The first extension addresses the problem of rounding errors in CLP($\Re$). We represent $\Re$eals with floating-point intervals which are computed by outward rounding. The second extension increases the expressiveness of linear CLP($\Re$). Constants in linear expressions can now be intervals, which enables reasoning with imprecise model parameters (tolerances). Bounds (*sup* and *inf*) for individual variables are computed by the linear optimization via modified Simplex. Both extensions are implemented in a CLP shell — an adaptation of SICStus Prolog, which allows for easy and fast developments and modifications of CLP languages.

1 Introduction

The motivation for this work was the idea to use CLP($\Re$) for the simulation and diagnosis of analog circuits. First experiments [Mozetič *et al.*, 1991] showed that CLP($\Re$) has some advantages over classical simulation tools (like SPICE or Micro-CAP) since the same model can be used for both, simulation and diagnosis. CLP($\Re$) can locate potential soft faults, i.e., parameter values (resistors, capacitors) deviating from nominal values. Recently [Novak *et al.*, 1993] this was combined with the design-for-test (DFT) methodology [Soma, 1990] in order to be able to distinguish between previously indistinguishable faults. The idea of the DFT methodology is to introduce MOS switches into a circuit in order to increase its controllability and observability.

Experiments with the simulation and diagnosis of analog circuits very soon revealed the floating-point problems of CLP($\Re$). A potential solution of using CLP($\mathcal{Q}$) instead of CLP($\Re$) is not practical. In our particular experiments CLP($\mathcal{Q}$) turned out to be 100 times

slower then CLP($\Re$) and needed considerably more space. What we tried next, and what we describe in this paper, is to extend CLP($\Re$) by real intervals.

Real intervals were already incorporated in CLP languages, e.g., in BNR Prolog [Older and Vellino, 1990, Older and Vellino, 1992] and in ICLP($\Re$) [Lee and van Emden, 1992]. The main motivation there is the control of rounding errors, and the ability to resolve nonlinear constraints. The mechanism for doing this is a new type of variables called an *interval*. A variable is initialized to an interval which subsequently narrows as more constraints on the variable are imposed. The constraint satisfaction mechanism for narrowing is based on local constraint propagation. It is non-symbolic and resembles a classical iterative numerical approximation technique.

In contrast, existing CLP($\Re$) [Jaffar and Michaylov, 1987] implementations are limited to systems of linear (in)equations and use symbolic (in)equation solving techniques. Here, variables which figure in inequalities can be *interpreted* as intervals. Their bounds (*sup* and *inf*) are not explicit, but can be computed by the Simplex optimization.

We introduce an extension to CLP($\Re$) where *generalized constants* can be used. A generalized constant is an ordered pair of floating-point numbers, defining the lower and upper bound of an interval which contains the original constant. These intervals are not narrowed during the computation — their ranges, however, have an effect on the variables. Originally ground variables might become just constrained (bound between *sup* and *inf*), and originally constrained variables might get looser bounds. The motivation for having generalized constants is twofold:

- The representation of $\Re$eals by floating-point intervals in order to cure the rounding problems. These generalized constants cannot be specified by the user — they are introduced by the low level numerical evaluator as a result of *outward rounding*.
- The increased expressive power of linear CLP($\Re$) by retaining the advantages of symbolic techniques. In this case, the generalized constants are introduced by the user in order to model parameter tolerances.

2 Outward rounding

The CLP shell [Holzbaur, 1992] we are using for our experiments is an extension of SICStus Prolog [Carlsson and Widen, 1991]. Various CLP instances themselves are implemented in Prolog and coded in a rather modular fashion. This coding discipline, together with the relative ease of applying partial evaluation to declarative programs, makes it possible to replace the numeric evaluator which works at the bottom of every CLP($\Re$, $\mathcal{Q}$) implementation. In fact, our CLP($\Re$) and CLP($\mathcal{Q}$) implementations share 99% of the code. The only difference besides some special case analyses that stem from the finite precision of floating-point numbers, is the numeric evaluator which computes with floating-point numbers in the case of CLP($\Re$), and with normalized rational numbers in the case of CLP($\mathcal{Q}$).

We implemented an outward rounding evaluator which returns a pair of floating-point numbers for each arithmetic expression involving (simple of generalized) constants. The pair encloses the actual results of the computation as tight as possible within the space of (hardware) representable floating-point numbers.

Although this evaluator is specified as a Prolog predicate, there is no call overhead during the execution of the resulting system, because we unfold references to the evaluator predicate. This is such a simple instance of partial evaluation, that we have no need for the power of a general partial evaluator. This leads us to unfolding efficiency — which is no issue after all because this process only takes place once when the CLP system is compiled.

3 Generalized constants

Replacing scalars, i.e., constants and coefficients in linear equations, by pairs of constants denoting lower and upper bound, allows one to model additional properties of parameters such as tolerances. This can be modelled directly in CLP($\Re$). Take, for example, the following system of three equations:

$$\begin{aligned} X_1 &= a \\ X_2 &= bX_1 + c \\ X_3 &= dX_1 + eX_2 + f \end{aligned}$$

Assume that a and b are generalized constants with given bounds:

$$\begin{aligned} X_1 &= [a_1, a_2] \\ X_2 &= [b_1, b_2]X_1 + c \\ X_3 &= dX_1 + eX_2 + f \end{aligned}$$

Each factor with a generalized coefficient can be replaced by a fresh variable and two inequalities:

$$\begin{aligned} X_2 &= X_1' + c \\ X_3 &= dX_1 + eX_2 + f \\ & \quad a_1 \leq X_1 \leq a_2 && (1) \\ X_1 \geq 0 \quad &: \quad b_1X_1 \leq X_1' \leq b_2X_1 && (2) \\ X_1 < 0 \quad &: \quad b_1X_1 \geq X_1' \geq b_2X_1 && (3) \end{aligned}$$

A generalized coefficient requires a case analysis (2 or 3). This can be executed directly by CLP($\Re$) (through backtracking), but the complexity is of course exponential in the number of generalized coefficients.

We propose a two pass solution. In the first pass, mean values of generalized constants are taken, yielding ground or (in general) constrained solutions for the variables:

$$\begin{aligned} \bar{X_1} &= \frac{a_1 + a_2}{2} \\ \bar{X_2} &= \frac{b_1 + b_2}{2}\bar{X_1} + c \\ \bar{X_3} &= d\bar{X_1} + e\bar{X_2} + f \end{aligned}$$

In the second pass, these mean solutions are used to resolve the cases (2 or 3), and looser bounds on the original variables are computed. This two-pass technique is effectively realized by the *freeze* mechanism [Carlsson and Widen, 1991], by delaying (2, 3) on $\bar{X_1}$ until it is sufficiently constrained. The technique is sound but incomplete. In general, a solution to the system of linear equations with generalized constants yields a *set* of bounds (interpreted as intervals) for each variable. The proposed technique finds just one interval, expanded around the ground solution, computed in the first pass from the mean values.

The question arises: "Why shouldn't one use the low level outward rounding evaluator to deal with all generalized constants (needed internally and introduced by the user) ?" The evaluator works under the assumption that terms submitted for evaluation are independent, even when several represent the same constant. This is the well know 'variable (constant) identity' problem. As a consequence, the tightness of the computed bounds is suboptimal. Therefore, to make these identities explicit, fresh variables with associated inequalities are introduced.

4 An example

We illustrate various interval handling techniques by a simple example of two resistors in a series (Figure 1). Both resistors and given voltages have tolerances. The goal is to compute the voltage and current ranges at the node between the resistors.

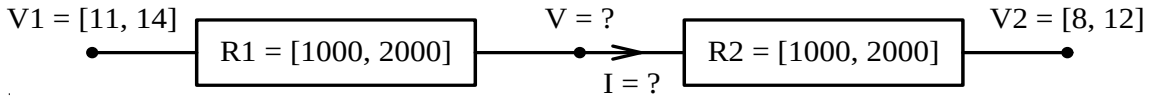

Figure 1: Two resistors with tolerances in a series.

The following CLP($\Re$) program specifies the circuit:

model(V, I) $\leftarrow$
V1 = [11,14]
V2 = [8,12],
resistor([1000,2000], I, V1, V),
resistor([1000,2000], I, V, V2).

resistor(R, I, V1, V2) $\leftarrow$ *R*I = V1 − V2.*

A variable is represented by a pair $v(\bar{X}, X)$, where $\bar{X}$ is computed from mean values of generalized constants, and X is bound between lower and upper bounds of generalized constants. The following code specifies the assignment of a generalized constant to a variable, and the multiplication of a generalized constant by a variable:

[A,B] = v($\bar{X}$,X) $\leftarrow$
$\bar{X} = \frac{A+B}{2}$,
$A \leq X,\ X \leq B.$

*[A,B] * v($\bar{X}$,X) = v($\bar{Y}$,Y)* $\leftarrow$
$\bar{Y} = \frac{A+B}{2} * \bar{X}$,
freeze($\bar{X}$*, bounds(* $\bar{X}$*, A, B, X, Y)).*

bounds($\bar{X}$*, A, B, X, Y)* $\leftarrow$ $\bar{X} \geq 0$,
*A*X* $\leq$ *Y, Y* $\leq$ *B*X.*
bounds($\bar{X}$*, A, B, X, Y)* $\leftarrow$ $\bar{X} < 0$,
*A*X* $\geq$ *Y, Y* $\geq$ *B*X.*

The multiplication of a generalized constant to a variable is delayed until $\bar{X}$ is ground. This reduces backtracking, but misses some solutions. The correct (but less efficient) definition of the *bounds* predicate should test the sign of X instead of $\bar{X}$.

The first query illustrates how the outward rounding is used to compute the *mean solution*, i.e., I and V are computed from the mean values of generalized constants. Instead of a single floating-point number, the result is a pair of floating-point numbers which contain the $\Re$eal solution:

$\leftarrow$ *I = (12.5 − 10)/(1500 + 1500),*
*V = 12.5 − 1500*I.*

I = [0.0008333333333333328, 0.0008333333333333335],
V = [11.249999999999999, 11.250000000000002]

The same mechanism applied to the actual generalized constants (instead of their mean values) yields a *naive block*, i.e., an interval in two dimensions. However, due to the 'constant identity' problem the block contains a considerable number of impossible solutions (Figure 2):

$\leftarrow$ *I = ([11,14] – [8,12])/([1000,2000] + [1000,2000]),*
*V = [11,14] – [1000,2000]*I.*

I = [–0.0005000000000000001, 0.0030000000000000001],
V = [4.9999999999999992, 15.000000000000002]

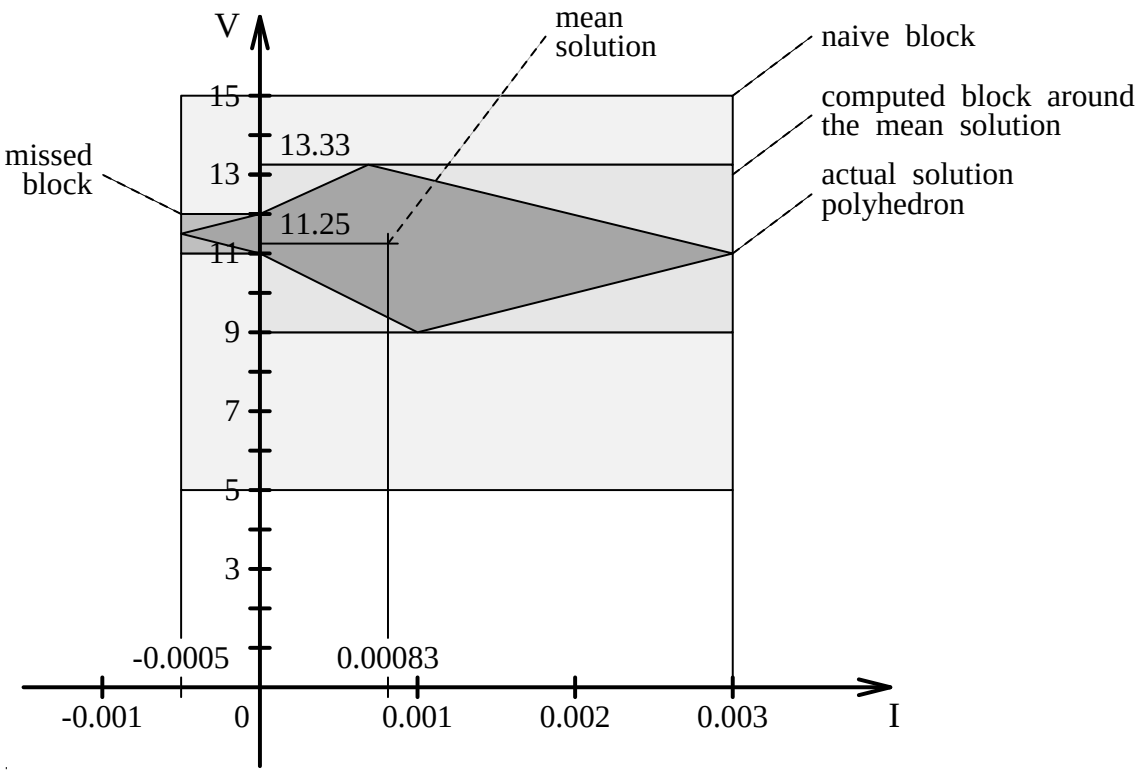

Figure 2: Interval ranges for the two resistors example.

If we consider the generalized constants then the *actual solution polyhedron* is determined by a set of inequalities. Through backtracking, CLP($\Re$) returns two solutions, depending whether $I \geq 0$ or $I < 0$:

$\leftarrow$ *model(V, I).*

I $\geq$ *0,*
I $\geq$ *0.0055 – 0.0005*V,*
I $\leq$ *–0.008 + 0.001*V,*
I $\leq$ *0.014 – 0.001*V,*
I $\geq$ *–0.006 + 0.0005*V ? ;*

I $<$ *0,*
I $\geq$ *–0.012 + 0.001*V,*
I $\geq$ *0.011 – 0.001*V*

It is convenient to represent the polyhedron by the smallest block which contains it. In CLP($\Re$) this can be done by computing *inf* and *sup* of each variable involved:

$\leftarrow$ *model(V, I),*
inf(V, Vinf), sup(V, Vsup),
inf(I, Iinf), sup(I, Isup).

$I \geq 0$,
...
$Iinf = 0$, $Isup = 0.003$,
$Vinf = 9$, $Vsup = 13.333333333333334$? ;

$I < 0$,
...
$Iinf = -0.0005$, $Isup = 0$,
$Vinf = 11$, $Vsup = 12$

The proposed two pass technique (with mean values and *freeze*) computes in the first pass the mean solutions, and in the second pass the *block around the mean solution* (Figure 2). The second solution (for $X < 0$, in the *missed block*) is not computed, since there is no backtracking.

5 Discussion

From the CLP($\Re$) solver's point of view the proposed technique amounts to an additional workload because of the two inequalities added for each constant or coefficient being generalized. While we are pleased with the additional modelling power of our system, we are not satisfied with the overall performance. Our current explanation is the following.

We decide the satisfiability of systems of inequalities by an incremental version of the Simplex [Dantzig, 1963] algorithm. The current version does not take advantage of special properties of the submitted inequalities.

One such property is the dimension (= number of variables) of the inequalities. There are variants of the Simplex algorithm which perform the costly basis transformations only for equations in more than one variable, i.e., simple upper and lower bounds on variables are treated specially (bounded variable linear programs [Murty, 1976]). Note that this restriction on the syntactical form of inequalities is local in the sense that it does not restrict the number of variables in the other inequalities of a system. This is in contrast to the global restrictions imposed by TVPI algorithms [Cohen and Megiddo, 1991].

Special treatment of inequalities of low dimension also facilitates the detection of redundant inequalities. For simple bounds this is an evaluable test, for inequalities in more dimensions one can (sometimes) detect syntactic redundancy by sorting [Lassez *et al.*, 1989] or hashing [Pugh, 1991]. In general, however, one has to run linear optimization programs to detect subsumption.

Therefore we plan to incorporate specializations for inequalities of low dimension into the decision algorithm for inequalities.

Acknowledgements

The authors acknowledge the support of the Austrian "Fonds zur Förderung der wissenschaftlichen Forschung" under grant P9426-PHY. Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Science and Research.

References

[Carlsson and Widen, 1991] Carlsson, M., Widen, J. SICStus Prolog user's manual. Swedish Institute of Computer Science, Kista, Sweden, 1991.

[Cohen and Megiddo, 1991] Cohen, E., Megiddo, N. Improved algorithms for linear inequalities with two variables per inequality. IBM Research Report, RJ 8187 (75146), 1991.

[Dantzig, 1963] Dantzig, G.B. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ, 1963.

[Holzbaur, 1992] Holzbaur, C. DMCAI CLP reference manual. Report TR-92-24, Austrian Research Institute for Artificial Intelligence, Vienna, Austria, 1992.

[Jaffar and Michaylov, 1987] Jaffar, J., Michaylov, S. Methodology and implementation of a CLP system. *Proc. 4th Intl. Conf. on Logic Programming*, pp. 196-218, Melbourne, Australia, MIT Press, 1987.

[Lassez *et al.*, 1989] Lassez, J.L., Huynh, T., McAloon, K. Simplification and elimination of redundant linear arithmetic constraints. *Proc. North American Conf. on Logic Programming 1989*, pp. 35-51, Cleveland, MIT Press, 1989.

[Lee and van Emden, 1992] Lee, J.H.M, van Emden, M.H. Adapting CLP($\Re$) to floating-point arithmetic. *Proc. Intl. Conf. on Fifth Generation Computer Systems*, pp. 996-1003, ICOT, Tokyo, 1992.

[Mozetič *et al.*, 1991] Mozetič, I., Holzbaur, C., Novak, F., Santo-Zarnik, M. Model-based analogue circuit diagnosis with CLP($\Re$). *Proc. 4th Intl. GI Congress*, pp. 343-353, Munich, Springer-Verlag, 1991.

[Murty, 1976] Murty, K.G. *Linear and Combinatorial Programming.* Wiley, New York, 1976.

[Novak *et al.*, 1993] Novak, F., Mozetič, I., Santo-Zarnik, M., Biasizzio, A. Enhancing design-for-test for active analog filters by using CLP($\Re$). *Analog Integrated Circuits and Signal Processing*, to appear, 1993.

[Older and Vellino, 1990] Older, W., Vellino, A. Extending Prolog with constraint arithmetic on real intervals. *Proc. Canadian Conf. on Electrical and Computer Engineering*, 1990.

[Older and Vellino, 1992] Older, W., Vellino, A. Constraint arithmetic on real intervals. In *Constraint Logic Programming: Selected Research* (F. Benhamou, A. Colmerauer, Eds.), MIT Press, to appear, 1993.

[Pugh, 1991] Pugh, W. The Omega test: a fast and practical integer programming algorithm for dependence analysis. *Proc. of Supercomputing*, 1991.

[Soma, 1990] Soma, M. A design-for-test methodology for active analog filters. *Proc. IEEE Intl. Test Conf. 1990*, pp. 183-192, Washington D.C., IEEE, 1990.