LOP '92 :- ! Rusava, 27. 1. - 1. 2. 1992

Efficient implementation of a CLP(R) system

Petr Mejzlík

Institute of Computer Science
Masaryk University
Burešova 20
CS – 601 77 Brno, Czechoslovakia
mejzlik@cspuni12.bitnet

Abstract: Constraint Logic Programming language CLP($\mathbb{R}$) integrates logic programming and handling with constraints on real terms. CLP($\mathbb{R}$) allows easy writing of declarative programs for hypothetical reasoning and complex decisions based on results of floating–point computations.
In this paper, an implementation of CLP($\mathbb{R}$) within a Prolog compiler, is described. The main aim of the implementation is to bring efficiency of CLP($\mathbb{R}$) programs nearer to programs written in procedural languages. A new incremental Simplex method is used to handle with linear constraints.

Keywords: Logic Programming, CLP, Simplex method

Introduction

Constraint Logic Programming (CLP) is a general scheme defining a class of languages for rule–based constraint programming. There are several implementations of CLP languages – CLP($\mathbb{R}$) [4], CHIP [2] and Prolog III [1]. With regard to constraints on real domain, CHIP and Prolog III permit linear rational terms while CLP($\mathbb{R}$) allows general real terms in constraints. This generality imposes a theoretical insufficiency of CLP($\mathbb{R}$) implementations since, by the theory, CLP($\mathbb{R}$) system should deal with all real numbers in the absolute precision.

In our implementation, complex constraints of a CLP($\mathbb{R}$) program are translated to a set of "standard" constraints in runtime. The constraints generated this way are pushed on the top of the CLP($\mathbb{R}$) solver stack. After every addition to the stack the solver tries to decide whether or not the system in the stack is solvable. When an unsolvability is detected, the solver activates a backtracking mechanism.

Communication between the Solver and Inference Engine

The CLP($\mathbb{R}$) system can be conceptionally decomposed into two parts – an inference engine and a solver. Our implementation uses modified inference engine of the Prolog compiler [5].

Generally, the inference engine sends a new constraint to the solver and the solver replies by a one–bit message expressing the solvability of a system of constraints extended by this new one. When the solver states an unsolvability, it returns to the same state as it was before the last sent constraint.

The solver also submits values of variables computed during the testing of solvability. In the opposite direction, the engine sends values of numerical variables determined without the use of the solver (i.e. the engine must recognize numerical and non–numerical variables).

Canonical Forms of Constraints

Complex constraints in a CLP($\mathbb{R}$) program are converted into an equivalent set of *canonical* constraints by an algorithm based on decomposition of the constraint to the corresponding syntactic tree.

The solver deals with the following canonical forms of constraints:

1) Linear equations and inequations
   $\mathbf{ax} = c, \quad \mathbf{ax} \leq c, \quad \mathbf{ax} \geq c.$
2) Equations of the type
   $t = x_1 x_2 \dots x_k, \quad t = x/y, \quad 0 \neq t.$
3) Equations of the type
   $t = \tau(s),$
   where τ is an unary real function $(\sin, \cos, \tan, \ln, \exp, \dots)$.
4) Binomial disjunctions of the type

$$(\mathbf{ax} \leq c \ \wedge \ \mathbf{bx} \leq d) \ \vee \ (\mathbf{ax} \geq c \ \wedge \ \mathbf{bx} \geq d) \tag{1}$$

Linear constraints of the type 1) are processed by a modified Simplex method separately from constraints of other types. A *nonlinear constraint* (i.e. a constraint of the type 2) or 3)) is only delayed while values of all its variables are unknown. Disjunctions of the type (1) are solved inside the linear solver.

Information represented by a nonlinear constraint is not used before all its right–side variables are known (suppose that the constraint is written in one of the above forms), with the following exceptions:

(i) If we have a constraint of the type $t = x_1 x_2 \dots x_k$ or $t = x/y$ and $x_i = 0$ for some i ($x = 0$ in the second case), we assign 0 to t.

(ii) In constraints of the type $t = \tau(s)$, the value of s can be computed from t when τ is an invertible function $(\exp, \ln, \dots)$.

Converting of Complex Constraints

In our implementation, the general syntax of constraints is

$$f(\mathbf{x}) \ \rho \ f'(\mathbf{x}) \tag{2}$$

where $f(\mathbf{x})$ and $f'(\mathbf{x})$ are real terms ($\mathbf{x}$ is a vector of real variables) and $\rho \in \{=, <, >, \leq, \geq, \neq\}$. Terms $f(\mathbf{x})$ and $f'(\mathbf{x})$ can contain real constants, variables from $\mathbf{x}$, predefined unary real functions and operators $+$, $-$, $\times$ and $\div$.

In the first step, the right side of (2) is subtracted from the left one. This reduces the conversion problem to constraints of the type

$$g(\mathbf{x}) \ \rho \ 0 \tag{3}$$

where $g(\mathbf{x})$ is a real term in variables from $\mathbf{x}$. Term $g(\mathbf{x})$ is transformed to the fractional form

$$\frac{\mathbf{a}(\mathbf{x},\mathbf{y})^T + c}{\mathbf{b}(\mathbf{x},\mathbf{y})^T + d} \tag{4}$$

where $\mathbf{a}$ and $\mathbf{b}$ are real vectors, $c, d \in \mathbb{R}$ and $\mathbf{y}$ is a vector of artificial variables created through the transformation. During the transformation, new constraints in variables from $\mathbf{x}$ and $\mathbf{y}$ are generated. The reduction algorithm is then recursively applied to the new constraints.

Example.
During the conversion of constraint

$$\frac{x_1 x_2}{\sin(\frac{x_1}{x_2})} < 0$$

the following constraints are generated:

$y_1 = x_1 x_2,\ y_2 = x_1/x_2,\ 0 \neq x_2,$

$y_3 = \sin(y_2),\ 0 \neq y_3,$

(now the original constraint corresponds to $\frac{y_1}{y_3} < 0$),

$(y_1 \leq 0 \ \wedge \ -y_3 \leq 0) \ \vee \ (y_1 \geq 0 \ \wedge \ -y_3 \geq 0),$

$0 \neq y_1.$

Linear Solver

The main part of the CLP($\mathbb{R}$) solver constitutes a linear solver based on algorithms of *incremental Simplex method* [6]. This method has been developed by a modification of a certain Simplex algorithm which uses so called *eta vectors* [3, 8].

Eta vectors are special square matrices of the type

$$\mathbf{E}_j = \begin{pmatrix} 1 & & \eta_{1l} & & \\ & \ddots & \vdots & & \\ & & \eta_{ll} & & \\ & & \vdots & \ddots & \\ & & \eta_{ml} & & 1 \end{pmatrix}$$

The change of the current basic solution to a new one can be expressed by an eta vector as

$$\mathbf{B}_{i+1}^{-1} = \mathbf{E}_i \mathbf{B}_i^{-1} \tag{5}$$

where $\mathbf{B}_i$ is the current basis and $\mathbf{B}_{i+1}$ is the new basis (it can be proved that there is eta vector $\mathbf{E}_i$ satisfying equation (5)). Therefore, the state of computation is fully defined by the sequence $\mathbf{E}_0, \mathbf{E}_1, \ldots, \mathbf{E}_i$. It is sufficient to store only matrices $\mathbf{E}_i$ in a compressed form instead of inverses of basis matrices $\mathbf{B}_i$.

The return to the previous state of computation (i.e. to the previous basic solution) is clear in the case when the linear system is known at the beginning of computation. In this case $\mathbf{B}_i^{-1}$ can be obtained by multiplying of matrix $\mathbf{B}_{i+1}^{-1}$ by $\mathbf{E}_i^{-1}$ from the left (eta vector $\mathbf{E}_i$ can be efficiently inverted due to its sparsity). Nevertheless, the linear solver of a CLP($\mathbb{R}$) system must also be able to restore efficiently the previous state of computation in the case when $\mathbf{B}_{i+1}$ is created from $\mathbf{B}_i$ by addition of a new variable column. Without any loss of generality we can suppose that the new column added to $\mathbf{B}_i$ is the last in $\mathbf{B}_{i+1}$. Then there exist $d \in \mathbb{R}$ and $\mathbf{b} \in \mathbb{R}^n$ (where n is the order of matrix $\mathbf{B}_i$) such that

$$\mathbf{B}_{i+1}^{-1} = \begin{pmatrix} \mathbf{B}_i^{-1} & \mathbf{0} \\ \mathbf{b} & d \end{pmatrix}$$

We can see that in this case in the change from $\mathbf{B}_i$ to $\mathbf{B}_{i+1}$ it is sufficient to store the last row of matrix $\mathbf{B}_{i+1}^{-1}$ that is in role of eta vector $\mathbf{E}_i$ stored when matrices $\mathbf{B}_i$ and $\mathbf{B}_{i+1}$ have the same order. The linear solver includes an extension of known algorithms for eta vectors computations using this data representation when
(order of $\mathbf{B}_{i+1}$) = (order of $\mathbf{B}_i$) + 1.

One part of the linear solver is a simple backtracking mechanism enabling to solve disjunctions of linear constraints (1). The linear solver also includes means for correcting the rounding errors [7]. The numerical stability of the solver was tested through randomly generated systems of about hundred constraints.

Acknowledgment

I am grateful to Luděk Matyska and David Toman for their support in the integration of the CLP($\mathbb{R}$) solver with Prolog compiler and valuable discussions on this work.

References

[1] A. Colmenauer. Opening the Prolog III universe. *Byte*, pages 177–182, August 1987.

[2] M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving a cutting–stock problem in constraint logic programming. In *Proc. 5th International Conference on Logic Programming*, pages 42–58. MIT Press, Cambridge, MA, 1988.

[3] S. I. Gass. *Linear Programming: Methods and Applications*. McGraw–Hill, New York, 5th edition, 1985.

[4] J. Jaffar and S. Michaylov. Methodology and implementation of a CLP system. In *Proc. 4th International Conference on Logic Programming*, pages 196–218. MIT Press, Cambridge, MA, 1987.

[5] L. Matyska and D. Toman. Prolog and WAM – A case study of an implementation. In *Proc. LOP'88*, pages 65–88, Ruprechtov, 1988. (Available also as the *Tech. Report*, ICS MU 1988).

[6] P. Mejzlík. An incremental Simplex method. in preparation.

[7] B. A. Murtagh. *Advanced Linear Programming: Computation and Practice.* McGraw–Hill, New York, 1981.

[8] J. Plesník, J. Dupačová, and J. Svobodová. *Lineárne programovanie.* Alfa, Bratislava, 1990. (in Slovak).