Integrating an algorithm for solving linear constraints in finite domains in the language CHIP*

Nicolas Beldiceanu†, Evelyne Contejean‡and Helmut Simonis†

CHIP is a constraint logic programming language originally designed and developed at ECRC [5], and further developed and marketed by COSYTEC. CHIP is able to solve among other constraints (on rationals and booleans), linear constraints over finite integer domains. In this case, its basic mechanism is the combination of constraint propagation and enumeration of the values in the domain associated with the given constraints. In this paper we present an original enumeration method based on an algorithm for solving systems of linear Diophantine equations due to E. Contejean and H. Devie [4]. The difficulty is to show that this enumeration method is compatible with constraint propagation, and that every solution is computed exactly once.

1 Contejean and Devie's Algorithm

This algorithm was first developed in order to solve linear equality constraints over natural numbers, but it can be very easily adapted in order to cope with linear equality and inequality constraints over finite domains. It is based on a certain enumeration of the potential solutions of a system, and termination is ensured by an adequate restriction on the search. This algorithm generalizes a previous algorithm due to Fortenbacher [2], which was restricted to the case of a single equation. A system S of p homogeneous linear Diophantine equations with q unknowns may be defined by its associated matrix $A = (a_{ij})_{1 \leq i \leq p,\ 1 \leq j \leq q}$ which has p rows and q columns:

$$S \quad \equiv \quad A\vec{x} = \vec{0}$$

We denote by e_j the j^{th} vector in the canonical basis of N^q: $e_j = (\ \underbrace{0, \ldots, 0}_{j-1 \text{ times}}\ , 1,\ \underbrace{0, \ldots, 0}_{q-j \text{ times}}\ )$

A solution of the system S is a tuple v which fulfills the following conditions:
1. v is in N^q, where N is the set of natural numbers,
2. v belongs to the kernel of the linear application defined by A with respect to the canonical basis.

The set of solutions of S is denoted by $\mathcal{S}$. The set of minimal solutions of S is denoted by $\mathcal{M}$. In the case of homogeneous systems, all tuples in $\mathcal{S}$ are linear combinations with natural coefficients of tuples in $\mathcal{M}$.

Fortenbacher's Algorithm

Assume that the system S is reduced to a single equation. It can then be solved by using Fortenbacher's algorithm [2] as follows: a directed acyclic graph rooted at the canonical vectors is searched

*This work is part of COLINE, a cooperative project between University Paris-Sud at Orsay and COSYTEC SA financed by French Ministère de la Recherche et de l'Espace, décision d'aide n^o 92 S 0600

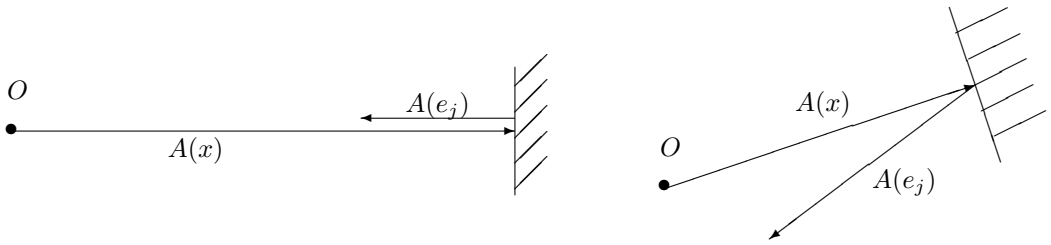
†COSYTEC, Parc Club Orsay Université, 4 rue Jean Rostand, 91893 ORSAY CEDEX, France

‡MPI Im Stadtwald, D-6600 Saarbrücken, Germany, contejea@mpi-sb.mpg.de

breadth-first (indeed, the e_j's are the smallest non-trivial potential solutions of S). If a node $(x_1, \ldots, x_q)$ is neither equal to or greater than an already encountered solution, then its sons are generated by increasing by 1 some of its components under the following restriction:
(1) *generate all tuples* $(x_1, \ldots, x_{j-1}, x_j + 1, x_{j+1}, \ldots, x_q)$ *such that* $A(x_1, \ldots, x_q) \times Ae_j < 0$.
The constructed graph contains all minimal solutions of S, that is the set $\mathcal{M}$, and is finite.

Generalization

In a previous work [3, 4], we have presented a powerful generalization of Fortenbacher's algorithm for the case of an arbitrary number of equations. The geometric interpretation of the condition (1) is displayed on figure 1. Now this condition can easily be generalized to a p-dimension space where



Fortenbacher's restriction

$A(x),\ A(e_j)\ \in Z$

Contejean and Devie's generalized restriction

$A(x),\ A(e_j)\ \in Z^p$

Figure 1: Geometric interpretation.

p stands for the number of equations in S. The generalized algorithm is similar to Fortenbacher's algorithm, except that the condition in order to build the sons of a node x is now:
(p) *generate all tuples* $(x_1, \ldots, x_{j-1}, x_j + 1, x_{j+1}, \ldots, x_q)$ *such that the extremity of* $A(x + e_j)$ *lies in the affin half-space containing the origin, and delimited by the hyperplan orthogonal to* $A(x)$ *which contains the extremity of* $A(x)$.
This can also be expressed in a very simple analytic way: $A(x_1, \ldots, x_q) \cdot Ae_j < 0$, where $\cdot$ is the scalar product of vectors. The generalized version is terminating and complete for the case of systems of linear homogeneous Diophantine equations.

From a graph to a forest

There are some redundancies since some tuples are computed several times, as sons of several distincts fathers. This can be fixed easily by using "frozen components". Assume that for each node x occurring in the graph built by the algorithm, there is a total (arbitrary) ordering $\prec_x$ on its sons. If a node x has two distinct sons $x + e_{j_1}$, and $x + e_{j_2}$ such that $x + e_{j_1} \prec_x x + e_{j_2}$, then the j_1-th component is frozen in the sub-graph rooted at $x + e_{j_2}$: it cannot be increased any more, even if the geometrical condition expressed by the scalar product is satisfied. This method is still complete, and builds a sub-forest of the original graph. Here is the part of the algorithm which generates the sons of a node x:

sons($\vec{x}$) := son-iter($\vec{x}$, first-index($\vec{x}$))
son-iter($\vec{x}$, i) := **if** frozen-component($\vec{x}$, i)
then if not(last-component($\vec{x}$, i))
then son-iter($\vec{x}$, next-component($\vec{x}$, i))
else
else if $A(\vec{x}) \cdot A(e_i) < 0$
then generate-a-node($x + e_i$)
else;
if not(last-component($\vec{x}$, i))
then son-iter(freeze-component(i, $\vec{x}$), next-component($\vec{x}$, i))

2 Integration of the algorithm in CHIP

In the case of finite domains, the termination is no longer a problem, and the above method can be used for enumerating all solutions. Assume that the linear constraints $A(\vec{x}) = \vec{0}$ has to be solved over the finite domain $\vec{D} = D_1 \times \ldots \times D_q$. The constraint propagation is combined with the above enumeration procedure yielding the following part of the algorithm for generating the sons of a node x:

sons($\vec{x} \in \vec{D}$) := son-iter($\vec{x} \in \vec{D}$, first-index($\vec{x} \in \vec{D}$))
son-iter($\vec{x} \in \vec{D}$, i) := **if** frozen-component($\vec{x} \in \vec{D}$, i)
then if not(last-component($\vec{x} \in \vec{D}$, i))
then son-iter($\vec{x} \in \vec{D}$, next-component($\vec{x} \in \vec{D}$, i))
else
else if is-sol($min(\vec{D})$) **or** $A(min(\vec{D})) \cdot A(e_i) < 0$
then generate-and-propagate($\vec{x} \in \vec{D} \& x_i > min(D_i)$)
else;
if not(last-component($\vec{x} \in \vec{D}$, i))
then if propagate($\vec{x} \in \vec{D} \& x_i = min(D_i)$) = $\vec{x} \in \vec{D} \& x_i = min(D_i)$
then son-iter($\vec{x} \in \vec{D} \& x_i = min(D_i)$, next-component($\vec{x} \in \vec{D}$, i))
else son-iter($\vec{x} \in \vec{D} \& x_i = min(D_i)$, first-index($\vec{x} \in \vec{D} \& x_i = min(D_i)$, i))

This enumeration procedure is complete and experimentation is currently under way.

References

[1] Alexandre Boudet, Evelyne Contejean, and Hervé Devie. A new AC-unification algorithm with a new algorithm for solving diophantine equations. In *Proc. 5th IEEE Symp. Logic in Computer Science, Philadelphia*, June 1990.

[2] M. Clausen and A. Fortenbacher. Efficient solution of linear diophantine equations. *Journal of Symbolic Computation, 8 (1 & 2)*, pages 201–216, 1989.

[3] Evelyne Contejean and Hervé Devie. Solving systems of linear diophantine equations. In *Proc. 3rd Workshop on Unification, Lambrecht, Germany.* University of Kaiserslautern, June 1989.

[4] Evelyne Contejean and Hervé Devie. Résolution de systèmes linéaires d'équations diophantiennes. *Comptes-Rendus de l'Académie des Sciences de Paris*, 313:115–120, 1991. Série I.

[5] M. Dincbas, P. Van Hentenryck, Helmut Simonis, A. Aggoun, T. Graf, and F.Berthier. The constraint logic programming language CHIP. In *Proc. Int. Conf. on Fifth Generation Computer Systems FGCS-88*, pages 693–702, 1988.