Specialising Finite Domain Programs Using Polyhedra

Jacob M. Howe and Andy King

Computing Laboratory
University of Kent, Canterbury, CT2 7NF, UK
{J.M.Howe, A.M.King}@ukc.ac.uk

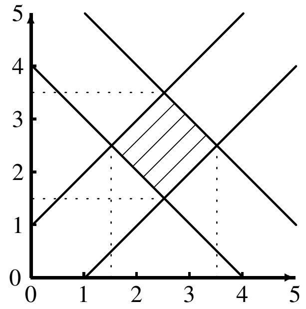
Abstract. A procedure is described for tightening domain constraints of finite domain logic programs by applying a static analysis based on convex polyhedra. Individual finite domain constraints are over-approximated by polyhedra to describe the solution space over n integer variables as an n dimensional polyhedron. This polyhedron is then approximated, using projection, as an n dimensional bounding box that can be used to specialise and improve the domain constraints. The analysis can be implemented straightforwardly and an empirical evaluation of the specialisation technique is given.

1 Introduction

Finite domain constraint logic programs classically have two components: a constraint component and a generate component. The constraint component posts to the store constraints which characterise the problem and define the search space. The generate component systematically enumerates the search space with a labelling strategy (such as fail first). Tightening the constraints, for example the domain constraints that bound the values of the variables, reduces the search space and thereby speeds up the program.

In order to reduce the search space, finite domain constraint solvers propagate constraints on the values that can be taken by the variables. Constraint propagation does not necessarily have to be applied with labelling and many solvers, for example the ECL^{i}PS^e and SICStus finite domain solvers, can prune the values of variables before any labelling is applied. This paper describes in detail and empirically evaluates one technique for performing constraint propagation at compiletime through program specialisation.

The analysis in this paper is founded on classic work on polyhedral approximation [5], [6]. Finite domain constraints are interpreted as relations over sets of points. These constraints are over approximated and represented as a (possibly unbounded) polyhedron. The intersection of polyhedra corresponds to composing constraints. Projection onto an integer grid gives (low-valency) domain constraints that can be added to the program without compromising efficiency. The main technique for propagating constraints in finite domain solvers is by bound propagation. This involves substituting known variable bounds into linear constraints to give new variable bounds. The polyhedral analysis described here is a stronger compiletime technique than bound propagation; compiletime bound propagation over linear finite domain constraints is subsumed by the technique described in this paper. The example in Figure 1 illustrates that polyhedral analysis can give considerably tighter approximations than those resulting from




```
:- use_module(library(clpfd)).
main:-
      domain([X, Y], 0, 6),
      Y#>=X-1,
      Y#=<X+1,
      Y#>=4-X,
      Y#=<6-X.
```

Fig. 1. The polyhedron represented by $\{y \geq x-1, y \leq x+1, y \geq 4-x, y \leq 6-x\}$ with variable domains $x \in [0,6], y \in [0,6]$.

bound propagation. In this example, projection onto each of the variables gives bounds $3/2 \leq x \leq 7/2, 3/2 \leq y \leq 7/2$. Tightening to integers defines the finite domain solution set $x \in [2,3], y \in [2,3]$, which can be used to specialise the domain constraints of the original program to `domain([X, Y], 2, 3)`. Bound propagation does not tighten the variable bounds at all.

The polyhedral analysis described in this paper develops the static analysis of constraint logic programs outlined in [14]. However, the analysis in this paper is specifically tailored to specialise finite domain programs. In particular, the analysis is designed to complement runtime constraint propagation techniques. As the example above illustrates, polyhedra capture deep inter-variable relationships which cannot always be traced in bound propagation. Note, however, that the technique is, to a certain extent, dependent on the data being present in the program – a static analysis cannot reason about runtime data. This paper makes the following contributions:

- it presents a deterministic algorithm (not involving labelling) based on polyhedra for refining domain constraints and it shows that the analysis can be easily implemented using constraint solving machinery;
- it shows how interval and polyhedral approximating techniques can be combined to reason about non-linear constraints;
- the analysis and the associated program transformation are shown to be correct;
- an empirical study and evaluation of the technique applied to SICStus finite domain programs is given. The analysis can significantly improve the speed of programs (sometimes by several orders of magnitude);
- applying the analysis through specialisation means that the solver does not need to be modified. Specialisation never impedes built-in constraint propagation techniques and comes with a no slow down guarantee. Moreover, the improved domain constraints often interact with built-in constraint propagation techniques resulting in further pruning. Interestingly, the analysis can be interpreted as a compiletime solution to combining constraint solvers.

The structure of the paper is as follows: section 2 works through an example program to illustrate the way in which the analysis works and its power; section 3 formalises the analysis in terms of abstract interpretation; section 4 describes the various mathematical techniques utilised in the analysis; section 5 compares the approach taken by this paper with bound propagation; section 6 works through another example program to illustrate all of the techniques introduced in the paper; section 7 describes the implementation of the analysis and gives the results of its application to some benchmark programs; section 8 reviews related work; section 9 concludes and outlines future work.

2 Example: Magic Square

This example illustrates the approach taken by this analysis, as well as its power relative to compiletime bound propagation.

The magic square puzzle takes a three by three grid and the numbers one to nine and sets the challenge of placing the numbers in the grid so that all of the rows, columns and diagonals sum to the same number. The solutions are ordered so as to reduce the number of solutions identical up to symmetry which can be found. A SICStus finite domain program to solve this problem is:

```
:- use_module(library(clpfd)).
square(A, B, C, D, E, F, G, H, I):-
        domain([A, B, C, D, E, F, G, H, I], 1, 9),
        all_different([A, B, C, D, E, F, G, H, I]),
        A#<C, A#<G, A#<I, %symmetry constraints
        A+B+C #= D+E+F, A+B+C #= G+H+I,
        A+B+C #= A+D+G, A+B+C #= B+E+H, A+B+C #= C+F+I,
        A+B+C #= A+E+I, A+B+C #= C+E+G,
        labeling([], [A, B, C, D, E, F, G, H, I]).
```

(In SICStus, `domain(List, Inf, Sup)` abbreviates `Inf#=<X, X#=<Sup`, for each variable `X` in `List`.) The finite domain constraints in this program are approximated by a polyhedron (each constraint is interpreted as a non-strict inequality with rational coefficients, these inequalities define the polyhedron). The `all_different` constraint cannot be captured in an informative way by a polyhedron, hence is ignored. The finite domain constraints are abstracted to the polyhedron defined by the following linear inequalities (an equality can be understood as a pair of inequalities):

$$
\begin{array}{lll}
1 \leq A, B, C, D, E, F, G, H, I \leq 9 & & \\
A \leq C - 1 & A \leq G - 1 & A \leq I - 1 \\
A + B + C = D + E + F & A + B + C = G + H + I & \\
A + B + C = A + D + G & A + B + C = B + E + H & \\
A + B + C = C + F + I & A + B + C = A + E + I & \\
A + B + C = C + E + G & &
\end{array}
$$

The above inequalities define a polyhedron in nine (the number of variables) dimensional rational space. Projection onto each variable will give rational bounds on those variables. The result of this is as follows:

$$
\begin{array}{ccc}
3/2 \leq A \leq 11/2 & 4 \leq B \leq 8 & 7/2 \leq C \leq 15/2 \\
5 \leq D \leq 9 & 3 \leq E \leq 7 & 1 \leq F \leq 5 \\
5/2 \leq G \leq 13/2 & 2 \leq H \leq 6 & 9/2 \leq I \leq 17/2
\end{array}
$$

A specialised finite domain program is obtained by reinterpreting these new rational bounds as finite domain bounds, by tightening to integer values. The constraint `domain([A, ..., I], 0, 9)` is replaced in the program by the finite domain constraints given below. The bounds in the left column below are those obtained by the above procedure, those on the right are those that SICStus finds by bound propagation.

```
%Polyhedral                    %Bound Propagation
2 #=< A, A #=< 5,              1 #=< A, A #=< 8,
4 #=< B, B #=< 8,              1 #=< B, B #=< 8,
4 #=< C, C #=< 7,              2 #=< C, C #=< 9,
5 #=< D, D #=< 9,              2 #=< D, D #=< 9,
3 #=< E, E #=< 7,              1 #=< E, E #=< 9,
1 #=< F, F #=< 5,              1 #=< F, F #=< 9,
3 #=< G, G #=< 6,              2 #=< G, G #=< 9,
2 #=< H, H #=< 6,              1 #=< H, H #=< 9,
5 #=< I, I #=< 8,              2 #=< I, I #=< 9,
```

Notice that the propagation of constraints by the polyhedral method is better than that of bound propagation. That the improvement is a large one can be seen by calculating the number of points in each of the search spaces. The finite domain which results from the polyhedral analysis has 8×10^5 points, whereas the domain resulting from bounds propagation has approximately 1.9×10^8 points, nearly 240 times larger a search space.

3 Formalised Analysis

This section formalises both the analysis and the program transformation described in this paper, then states their correctness. Details and proofs can be found in [9].

3.1 Polyhedral Analysis

In order to have confidence in the analysis a mathematical justification is essential. The formalisation is an application of the s-approach detailed in [4] and is fairly dense and complicated. Thus, before giving the formal analysis, an informal overview of the remainder of the section is given, indicating where the operations described in section 4 are required. Abstract interpretation is used to connect a (concrete) ground semantics for finite domain constraint programs [11], [12] to an (abstract) s-semantics [4]. A Galois insertion links the concrete domain (the set of ground interpretations) and the abstract domain (the set of interpretations over constrained unit clauses). The concrete semantics is essentially the set of solutions for a given program. The abstract semantics (formulated in terms of a fixpoint) is an over-approximation of this set of solutions, with each predicate constrained by the conjunction of the constraints on its body atoms. The abstract operator approximates non-linear constraints as linear constraints. In order that the formalised analysis is the same as that implemented, the number of unit clauses is kept small by over-approximation in the form of a convex hull calculation. The termination of the fixpoint calculation is ensured by the use of a widening. The analysis is proved to be correct, as is the program transformation (which involves the use of projection with the fixpoint).

Concrete Domain For a (finite domain) program P, let Π denote the set of predicate symbols that occur in P and let Σ denote the set of integer ($\mathbb{Z}$) and function symbols that occur in P. Let D_{FD} be the set of finite trees over the signature Σ. Let R_{FD} be the set of constraint predicates. Let V be a countable set of variables. C_{FD} is the system of finite domain constraints generated from D_{FD}, R_{FD}, V and the function symbols. Elements of C_{FD} are regarded modulo logical equivalence and C_{FD} is ordered by entailment, $\models_{FD}$. $(C_{FD}, \models_{FD}, \wedge)$ is a (bounded) meet-semilattice with bottom and top elements $true$ and $false$. C_{FD} is closed under variable elimination and $\exists\{x_1, ..., x_n\}c$ (projection out) abbreviates $\exists x_1 \ldots \exists x_n.c$. $\overline{\exists} X\, c$ (projection onto) is used as a shorthand for $\exists(var(c) \setminus X)c$, where $var(o)$ denotes the set of variables occurring in the syntactic object o. The interpretation base for P is $B_{FD} = \{p(\bar{\mathbf{t}}) \mid p \in \Pi, \bar{\mathbf{t}} \in (D_{FD})^n\}$. The concrete domain is $(\mathcal{P}(B_{FD}), \subseteq, \cap, \cup)$, a complete lattice.

Abstract Domain Let D_{Lin} be the set of rational numbers, $\mathbb{Q}$. Let C_{Lin} be the system of linear constraints over D_{Lin}, V, the set of constraint predicates R_{Lin} and the function symbols. C_{Lin} is quotiented by equivalence and ordered by entailment, $\models_{Lin}$. $(C_{Lin}, \models_{Lin}, \wedge)$ is a (bounded) meet-semilattice and is closed under projection out, $\exists$, and projection onto, $\overline{\exists}$. Unit clauses have the form $p(\bar{\mathbf{x}}) \leftarrow c$ where $c \in C_{Lin}$. Equivalence on clauses, $\equiv$, is defined as follows: $(p(\bar{\mathbf{x}}) \leftarrow c) \equiv (p(\bar{\mathbf{x}}') \leftarrow c')$ iff $\overline{\exists} var(\bar{\mathbf{x}})c = \overline{\exists} var(\bar{\mathbf{x}})(c' \wedge (\bar{\mathbf{x}} = \bar{\mathbf{x}}'))$. The interpretation base for program P is $B_{Lin} = \{[p(\bar{\mathbf{x}}) \leftarrow c]_\equiv | p \in \Pi, c \in C_{Lin}\}$. Entailment induces an order relation, $\sqsubseteq$, on $\mathcal{P}(B_{Lin})$ as follows: $I \sqsubseteq I'$ iff $\forall [p(\bar{\mathbf{x}}) \leftarrow c]_\equiv \in I. \exists [p(\bar{\mathbf{x}}) \leftarrow c']_\equiv \in I'.c \models_{Lin} c'$. $\mathcal{P}(B_{Lin})$ ordered by $\sqsubseteq$ is a preorder. Quotienting by equivalence, $\equiv$, gives the abstract domain $(\mathcal{P}(B_{Lin})/\equiv, \sqsubseteq, \sqcup)$, a complete join-semilattice, where $\sqcup_{i=1}^{\infty}[I_i]_\equiv = [\cup_{i=1}^{\infty} I_i]_\equiv$.

Concretisation The concretisation map $\gamma : C_{Lin} \to C_{FD}$, interprets a linear constraint over the rationals as a finite domain constraint as follows:

$$\gamma\left(\sum_{i=1}^{m} \frac{n_i}{d_i} x_i \leq \frac{n}{d}\right) = \sum_{i=1}^{m} \frac{D.n_i}{d_i} x_i \leq \frac{D.n}{d}, \text{ where } D = d. \prod_{i=1}^{m} d_i$$

Note that the coefficients of $\gamma(c_{Lin})$ are in $\mathbb{Z}$. The abstraction map, $\alpha : C_{FD} \to C_{Lin}$ can be defined in terms of γ by $\alpha(c_{FD}) = \wedge\{c_{Lin} | c_{FD} \models_{FD} \gamma(c_{Lin})\}$. Observe that α, γ form a Galois insertion.

The concretisation map $\gamma : \mathcal{P}(B_{Lin})/\equiv \to \mathcal{P}(B_{FD})$ on interpretations is defined in terms of the concretisation map for constraints:

$$\gamma([I]_\equiv) = \{p(\bar{\mathbf{t}}) | [p(\bar{\mathbf{x}}) \leftarrow c]_\equiv \in I, (\bar{\mathbf{x}} = \bar{\mathbf{t}}) \models_{FD} \gamma(c)\}.$$

The abstraction map $\alpha : \mathcal{P}(B_{FD}) \to \mathcal{P}(B_{Lin})/\equiv$ is defined as follows:

$$\alpha(J) = [\{[p(\bar{\mathbf{x}}) \leftarrow c]_\equiv | p(\bar{\mathbf{t}}) \in J, \alpha(\bar{\mathbf{x}} = \bar{\mathbf{t}}) = c\}]_\equiv$$

Proposition 1 *α, γ on interpretations form a Galois insertion.*

Concrete Semantics The fixpoint semantics, $\mathcal{F}_{FD}$, is defined in terms of an immediate consequences operator $T_P^g : \mathcal{P}(B_{FD}) \to \mathcal{P}(B_{FD})$, defined by

$$T_P^g(I) = \left\{ p(\bar{\mathbf{t}}) \,\middle|\, \begin{array}{l} w \in P, w = p(\bar{\mathbf{x}}) \leftarrow c, p_1(\bar{\mathbf{x}}_1), ..., p_n(\bar{\mathbf{x}}_n), \\ p_i(\bar{\mathbf{t}}_i) \in I, (\bar{\mathbf{x}} = \bar{\mathbf{t}}) \models_{FD} \overline{\exists} var(\bar{\mathbf{x}})(\wedge_{i=1}^n (\bar{\mathbf{x}}_i = \bar{\mathbf{t}}_i) \wedge c) \end{array} \right\}$$

T_P^g is continuous, thus the least fixpoint exists and $\mathcal{F}_{FD}[\![P]\!] = lfp(T_P^g)$.

Abstract Semantics To define the immediate consequences operator for the abstract semantics, a special conjunction operator $\wedge_{FL} : C_{FD} \times C_{Lin} \to C_{Lin}$ is introduced. The operator $\wedge_{FL}$ is assumed to satisfy the property $c_{FD} \wedge \gamma(c_{Lin}) \models_{FD} \gamma(c_{FD} \wedge_{FL} c_{Lin})$. This operator allows the approximation of non-linear finite domain constraints.

The fixpoint semantics, $\mathcal{F}_{Lin}$, is defined in terms of an immediate consequences operator, $T_P^s : \mathcal{P}(B_{Lin})/\equiv \to \mathcal{P}(B_{Lin})/\equiv$, defined by $T_P^s([I]_\equiv) = [J]_\equiv$, where

$$J = \left\{ [p(\bar{\mathbf{x}}) \leftarrow c]_\equiv \,\middle|\, \begin{array}{l} w \in P, w = p(\bar{\mathbf{x}}) \leftarrow c', p_1(\bar{\mathbf{x}}_1), ..., p_n(\bar{\mathbf{x}}_n), \\ [w_i]_\equiv \in I, w_i = p_i(\bar{\mathbf{y}}_i) \leftarrow c_i, \\ \forall i.(var(w) \cap var(w_i) = \phi), \\ \forall i \neq j.(var(w_i) \cap var(w_j) = \phi), \\ c = c' \wedge_{FL} (\wedge_{i=1}^n ((\bar{\mathbf{x}}_i = \bar{\mathbf{y}}_i) \wedge c_i)) \end{array} \right\}$$

T_P^s is continuous, thus $lfp(T_P^s)$ exists. Since $\mathcal{P}(B_{Lin})/\equiv$ is a complete partial order, Kleene iteration [5] can be used to compute $\mathcal{F}_{Lin}[\![P]\!] = lfp(T_P^s) = \sqcup_{i=1}^{\infty} T_P^s \uparrow i$, where $T_P^s \uparrow 0 = \phi$ and $T_P^s \uparrow i+1 = T_P^s(T_P^s \uparrow i)$.

Space-Efficient Over-Approximation To keep the number of unit clauses in $T_P^s \uparrow k$ small, hence the fixpoint calculation manageable, $T_P^s \uparrow k$ is over-approximated by an interpretation I (that is, $T_P^s \uparrow k \sqsubseteq I$) containing at most one unit clause for each predicate symbol.

The join for the domain of linear constraints, $\vee : C_{Lin} \times C_{Lin} \to C_{Lin}$, is defined by $c_1 \vee c_2 = \wedge\{c \in C_{Lin} | c_1 \models_{Lin} c, c_2 \models_{Lin} c\}$. When the constraints are interpreted as defining polyhedra, the meet corresponds to the closure of the convex hull. The operator is lifted in stages to an operator on the abstract domain. First it is lifted to the interpretation base, $\vee : B_{Lin}^\perp \times B_{Lin}^\perp \to B_{Lin}^\perp$, where $B_{Lin}^\perp = B_{Lin} \cup \{\perp\}$, as follows:

$$\begin{array}{llll} [p(\bar{\mathbf{x}}) \leftarrow c_1]_\equiv \vee [p(\bar{\mathbf{x}}) \leftarrow c_2]_\equiv & = [p(\bar{\mathbf{x}}) \leftarrow c_1 \vee c_2]_\equiv & \\ [p(\bar{\mathbf{x}}) \leftarrow c_1]_\equiv \vee [q(\bar{\mathbf{y}}) \leftarrow c_2]_\equiv & = \perp & \text{if } p \neq q \\ [p(\bar{\mathbf{x}}) \leftarrow c]_\equiv \vee \perp & = [p(\bar{\mathbf{x}}) \leftarrow c]_\equiv & \\ \perp \vee [p(\bar{\mathbf{x}}) \leftarrow c]_\equiv & = [p(\bar{\mathbf{x}}) \leftarrow c]_\equiv & \end{array}$$

This in turn defines the unary function, $\vee : \mathcal{P}(B_{Lin})/\equiv \to \mathcal{P}(B_{Lin})/\equiv$, on the abstract domain given by $\vee([I]_\equiv) = [\cup_{w \in I}\{\vee_{u \in I}(w \vee u)\}]_\equiv$. Since for every $I \in \mathcal{P}(B_{Lin})/\equiv$, $T_P^s(I) \sqsubseteq \vee \circ T_P^s(I)$, it follows that $lfp(T_P^s) \sqsubseteq lfp(\vee \circ T_P^s)$. Hence $\vee$ does not compromise safety.

Termination of the Polyhedral Analysis As before, Kleene iteration can be used to compute $lfp(\vee \circ T_P^s)$. However, the chain of iterates $\vee \circ T_P^s \uparrow k$ may not stabilise in a finite number of steps. In order to obtain convergence, widening (a fixpoint acceleration technique) [5], is applied.

Given a standard widening on polyhedra [3], [5], [6] (or equivalently, on linear constraints), $\nabla : C_{Lin} \times C_{Lin} \rightarrow C_{Lin}$, a widening, $\nabla : B_{Lin}^{\perp} \times B_{Lin}^{\perp} \rightarrow B_{Lin}^{\perp}$, (where $B_{Lin}^{\perp} = B_{Lin} \cup \{\perp\}$) on the interpretation base is induced as follows:

$$
\begin{array}{llll}
[p(\overline{\mathbf{x}}) \leftarrow c_1]_{\equiv} & \nabla\ [p(\overline{\mathbf{x}}) \leftarrow c_2]_{\equiv} & = [p(\overline{\mathbf{x}}) \leftarrow c_1 \nabla c_2]_{\equiv} & \\
[p(\overline{\mathbf{x}}) \leftarrow c_1]_{\equiv} & \nabla\ [q(\overline{\mathbf{y}}) \leftarrow c_2]_{\equiv} & = \perp & \text{if } p \neq q \\
[p(\overline{\mathbf{x}}) \leftarrow c]_{\equiv} & \nabla\ \perp & = [p(\overline{\mathbf{x}}) \leftarrow c]_{\equiv} & \\
\perp & \nabla\ [p(\overline{\mathbf{x}}) \leftarrow c]_{\equiv} & = [p(\overline{\mathbf{x}}) \leftarrow c]_{\equiv} &
\end{array}
$$

This lifts to the abstract domain, $\nabla : \mathcal{P}(B_{Lin})/\!\equiv\ \times \mathcal{P}(B_{Lin})/\!\equiv\ \rightarrow \mathcal{P}(B_{Lin})/\!\equiv$

$$[I_1]_{\equiv} \nabla [I_2]_{\equiv} = [\cup_{w \in I_2} \{\vee_{u \in I_1} (w \nabla u)\}]_{\equiv}$$

3.2 Correctness of the Polyhedral Analysis

This section states the correctness of the analysis. That is, upward iteration of $\vee \circ T_P^s$, with widening, stabilises at an interpretation I with $lfp(T_P^g) \sqsubseteq \gamma(I)$. The result is a corollary of Proposition 13 in [5].

Proposition 2 *The upward iteration sequence of $\vee \circ T_P^s$ with widening ∇ is ultimately stable with limit I and I is safe, that is, $\vee \circ T_P^s(I) \sqsubseteq I$ and $lfp(T_P^g) \subseteq \gamma(I)$.*

3.3 Program Transformation and its Correctness

Once an upper approximation to $\mathcal{F}_{FD}[\![P]\!]$ is computed, it can be used to transform the program. This is done by projecting the convex polyhedron resulting from the fixpoint calculation onto each variable in turn, tightening this interval constraint to integer values and adding it to the initial program. The following theorem details the transformation and also asserts safety.

An auxiliary (partial) map, $\cdot^t : C_{Lin} \rightarrow C_{Lin}$, is defined in order to tighten bounds on variables to integer values, as follows: $c^t = u(c) \wedge l(c)$ where

$$u(c) = \begin{cases} x \leq \lfloor q \rfloor & \text{if } (x \leq q) = c \\ true & \text{otherwise} \end{cases}, \qquad l(c) = \begin{cases} x \geq \lceil q \rceil & \text{if } (x \geq q) = c \\ true & \text{otherwise} \end{cases}.$$

Theorem 1 *If $lfp(T_P^g) \subseteq \gamma([I]_{\equiv})$, then $\mathcal{F}_{FD}[\![P]\!] = \mathcal{F}_{FD}[\![P']\!]$, where*

$$P' = \left\{ w' \left| \begin{array}{l} w \in P, w = p(\overline{\mathbf{x}}) \leftarrow c, p_1(\overline{\mathbf{x}}_1), ..., p_n(\overline{\mathbf{x}}_n), \\ [w_i]_{\equiv} \in I, w_i = p_i(\overline{\mathbf{y}}_i) \leftarrow c_i, \\ \forall i.(var(w) \cap var(w_i) = \phi), \\ \forall i \neq j.(var(w_i) \cap var(w_j) = \phi), \\ c' = c \wedge (\wedge_{y \in var(w)} \gamma((\overline{\exists} y \wedge_{i=1}^n ((\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i) \wedge c_i))^t)), \\ w' = p(\overline{\mathbf{x}}) \leftarrow c', p_1(\overline{\mathbf{x}}_1), ..., p_n(\overline{\mathbf{x}}_n) \end{array} \right. \right\}$$

4 Computational Techniques

The analysis and program transformation strategy is parameterised by the operators $\wedge_{FL}$ and ∇. In this section instances of these operators are specified and algorithms for computing other operations, such as $\vee$ and $\overline{\exists}$, are presented.

In particular this section reviews some computational techniques for: calculating the convex hull of two n dimensional polyhedra; projecting an n dimensional polyhedra onto an m dimension space where $m < n$; widening chains of polyhedra; approximating non-linear constraints by polyhedra.

4.1 Projection

The analysis described in this paper requires a projection that takes as input a set of inequalities in n variables and outputs a set of inequalities in a subset of these variables. The output is such that all solutions of the original set of inequalities can be specialised to a solution of the new, and all solutions of the new set of inequalities represent partial solutions of the original. Less formally, projection is the calculation of the shadow cast by the polyhedron represented by the inequalities onto the space defined by the subset of variables. For example, the projection of a two dimension polyhedron onto the variable x is the shadow cast onto the x-axis when the polyhedron is lit from above.

In the implementation of the analysis given in this paper, projection is performed using Fourier-Motzkin variable elimination (see, for example, [10], [12], [13], [18]), as this is the algorithm used by SICStus. Fourier-Motzkin variable elimination takes a set of linear inequalities and eliminates variables one at a time until the only variable occurrences left are of those variables being projected onto. Inequalities are arranged so that the variable to be eliminated is on the lesser side of all inequalities in which it occurs. It will either have a positive or negative polarity. All possible ways of eliminating the variable from a pair of inequalities are explored, giving a new set of inequalities with one variable fewer. This is illustrated with the following simple example, projecting onto the single variable z:

$$\begin{array}{c} x+y \leq 4 \\ x-y \geq 6 \\ z \leq y \end{array} \overset{\text{shuffle}}{\rightsquigarrow} \begin{array}{c} x \leq 4-y \\ -x \leq -6-y \\ z \leq y \end{array} \overset{\text{eliminate}}{\rightsquigarrow} \begin{array}{c} 0 \leq -2-2y \\ z \leq y \end{array} \overset{\text{shuffle}}{\rightsquigarrow} \begin{array}{c} y \leq -1 \\ -y \leq -z \end{array} \overset{\text{eliminate}}{\rightsquigarrow} z \leq -1$$

4.2 Convex Hull

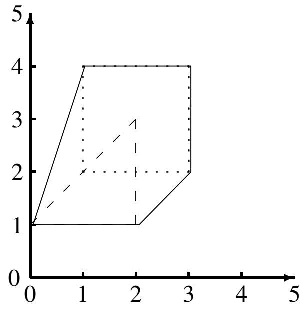
The convex hull of two polyhedra is the smallest polyhedron containing both polyhedra. The convex hull calculations are performed as in [3]. Polyhedra are represented as a set of linear inequalities. The convex hull, P_C, of two polyhedra, P_1 and P_2, is given by the following (where $\overline{\mathbf{x}}$ is a vector and A_i, B_i are matrices, together giving the linear inequalities that define the polyhedra):

$$P_1 = \{\overline{\mathbf{x}}_1 \in \mathbb{Q}^n \,|\, A_1\overline{\mathbf{x}}_1 \leq B_1\}, \qquad P_2 = \{\overline{\mathbf{x}}_2 \in \mathbb{Q}^n \,|\, A_2\overline{\mathbf{x}}_2 \leq B_2\}$$

$$P_C = \left\{ \overline{\mathbf{x}} \in \mathbb{Q}^n \,\middle|\, \begin{array}{l} \overline{\mathbf{x}} = \overline{\mathbf{y}}_1 + \overline{\mathbf{y}}_2 \ \wedge\ A_1\overline{\mathbf{y}}_1 \leq \sigma_1 B_1 \ \wedge\ A_2\overline{\mathbf{y}}_2 \leq \sigma_2 B_2 \\ \wedge\, \sigma_1 + \sigma_2 = 1 \ \wedge\ -\sigma_1 \leq 0 \ \wedge\ -\sigma_2 \leq 0 \end{array} \right\}$$

By projecting out $\sigma_1, \sigma_2, \overline{\mathbf{y}}_1, \overline{\mathbf{y}}_2$, that is, projecting onto $\overline{\mathbf{x}}$, the linear inequalities for the convex hull can be found. In this way, the convex hull calculation is reduced to variable elimination.

Example 1. Figure 2 lists a program giving rise to polyhedra that are a square and triangle. The triangle P_T and the square P_S are described below:



```
:- use_module(library(clpfd)).
p(X, Y):-
      -X+Y#=<1,
      X#=<2,
      1#=<Y
p(X, Y):-
      1#=<X, X#=<3,
      2#=<Y, Y#=<4.
```

Fig. 2. The Convex Hull of triangle P_T and square P_S

$$P_T = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \middle| \begin{pmatrix} -1 & 1 \\ 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \right\}, \; P_S = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \middle| \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} -1 \\ 3 \\ -2 \\ 4 \end{pmatrix} \right\}$$

Putting these together as in the definition, and then projecting out $\sigma_1, \sigma_2, \overline{\mathbf{y}}_1, \overline{\mathbf{y}}_2$, the convex hull, P_C, is found to be

$$P_C = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \middle| \begin{pmatrix} 0 & -1 \\ 1 & -1 \\ 1 & 0 \\ 0 & 1 \\ -3 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \\ 3 \\ 4 \\ 1 \end{pmatrix} \right\}$$

Observe that P_C describes the convex hull: a pentagon.

4.3 Approximating Non-Linear Constraints

A non-linear inequality cannot be accurately approximated by polyhedra. However, suppose that the non-linear inequality I describes a region R and that P is a polyhedron. The intersection $R \cap P$ can sometimes be approximated by a polyhedron P' such that $P' \subset P$. This problem arises in the analysis of finite domain programs that contain non-linear constraints. This section describes an algorithm for computing such a P' given non-linear inequality I and polyhedron P.

The following approximation technique arose from bound propagation algorithms for non-linear inequalities [13] and is outlined below:

1. I is rewritten to $(\wedge_{i=1}^{n} \prod X_i \leq \prod Y_i) \wedge (\wedge_{j=1}^{m} \sum Z_j \leq c_j)$ where X_i, Y_i, Z_j are variable multisets and c_j is a constant. For brevity the rewrite rules are omitted, instead a simple illustrative example is given. The non-linear inequality $z \leq x \times (u+v) \times y$ is rewritten to $z \leq x \times a \times y, a - u - v \leq 0, u + v - a \leq 0$, where a is a fresh variable.
2. In each product $\prod W_i$, where $W_i = \{w_1, ..., w_n\}$, every variable w_i has an upper bound, u_i, and a lower bound, l_i, which can be calculated by projecting P onto that variable (where $l_i, u_i \in \mathbb{Q} \cup \{+\infty, -\infty\}$). For every $k \in \{1, ..., n\}$, upper and lower bounds for the product $\prod W_i$ are computed by $w_k^u = w_k \cdot \prod_{i \in S} u_i$ and $w_k^l = w_k \cdot \prod_{i \in S} l_i$, where $S = \{1, ..., n\} - \{k\}$.
3. The upper and lower bounds on the products generate the following linear constraint for each non-linear inequality $\prod X_i \leq \prod Y_i$, where $X_i = \{x_1, ..., x_n\}, Y_i = \{y_1, ..., y_m\}$:

$$L_i = \wedge \{x_k^l \leq y_j^u |\, k \in \{1, ..., n\}, j \in \{1, ..., m\}\}$$

4. Finally the region $R \cap P$ is approximated by the polyhedron $P' = R' \cap P$ where R' is the polyhedron represented by

$$(\wedge_{i=1}^{n} L_i) \wedge (\wedge_{j=1}^{m} \sum Z_j \leq c_j)$$

Example 2. Consider the region $R = \{(x, y, z) | z \leq x \times y\}$ and the polyhedron $P = \{(x, y, z) | 1 \leq x, 2 \leq y \leq 4\}$. The region $R \cap P$ is approximated by a polyhedron P'. The non-linear inequality does not need to be rewritten as it is already in the required form. Projecting P onto x and y gives $1 \leq x \leq \infty$ and $2 \leq y \leq 4$. Then, $x \times y$ has upper bounds ∞, $4x$ and lower bounds y, $2x$. z has itself as upper and lower bounds. These generate the following linear inequalities $z \leq \infty, y \leq z, z \leq 4x, 2x \leq z$. Call the region generated by these inequalities R'. Then $P' = R' \cap P$.

The inequalities that arise assume the form $c_1 x \leq c_2 y$ rather than $c_1 x \leq c_2$. This is because if a tighter bound on x (or y) is later found, then the inequality $c_1 x \leq c_2 y$ can potentially tighten y (or x). This can only improve accuracy.

The analysis of non-linear constraints given here is an instance of the special conjunction operation, $\wedge_{FL}$, given in section 3: $R \wedge_{FL} P = P'$.

4.4 Widening

Widening is required to ensure that the fixpoint calculation will stabilise, that is, the polyhedra in the final two iterates coincide. Widenings for polyhedra can be found in [3], [5] and [6]. To keep the exposition reasonably self-contained, the [5] widening is detailed here.

Polyhedra are represented by sets of linear inequalities. If the previous iteration has produced polyhedron $P_k = \{\bar{\mathbf{x}} \in \mathbb{Q}^n \mid \wedge S_k\}$, where $S_k = \{I_1, ..., I_l\}$ and the current iteration has given polyhedron $P_{k+1} = \{\bar{\mathbf{x}} \in \mathbb{Q}^n \mid \wedge S_{k+1}\}$, where $S_{k+1} = \{J_1, ..., J_m\}$ (I_i and J_j are linear inequalities), then applying the widening results in the polyhedron given by the following set of linear inequalities:

$$\{I \in S_k | \wedge S_{k+1} \models I\} \cup \{J \in S_{k+1} | \exists I \in S_k. \wedge ((S_k - \{I\}) \cup \{J\}) = \wedge S_k\}.$$

Example 3. A smaller triangle $P_1 = \{(x, y)|y \leq x, x \leq 1, y \geq 0\}$, and a larger triangle $P_2 = \{(x, y)|y \leq 2x, x \leq 1, y \geq 0\}$ are widened to give the region $P_1 \nabla P_2 = \{(x, y)|x \leq 1, y \geq 0\}$. The inequality $y \leq x$ from P_1 is not satisfied by all points in P_2 and the other inequalities in P_1 are. The inequality $y \leq 2x$ from P_2 does not satisfy the swapping condition, and the other inequalities describing P_2 do.

5 Comparison With Bound Propagation

As noted above, the polyhedral analysis described in this paper subsumes compile-time bound propagation. Bound propagation is used in finite domain systems, such as ECL^{i}PS^e and SICStus. Good expositions of bound propagation can be found in [1] and [13]. A brief outline of the technique is given here.

Given any inequality, the known bounds for each of the variables occurring in the inequality are used to find possibly tighter bounds for these variables. One variable is chosen and the upper and lower bounds for the other variables are used to find a possible upper or lower bound for this chosen variable. If the bound calculated in this way is tighter than the previous known bound for that variable, this bound is adopted in place of the older, weaker one. This process can be repeated for each variable in the inequality. An equality can be treated as two inequalities. To give a very simple illustrative example consider the following two variable case:

$$y = x + 7, 0 \leq x \leq 3, 0 \leq y \leq 12.$$

Propagating the bounds on x into the inequalities involving x and y it is found that $7 \leq y \leq 10$, tighter bounds than previously.

Bound propagation can give good tightening of constraints. For example, bound propagation in the send more money problem (one of the example programs, see Table 1) actually gives the same results as the polyhedral analysis! However, there are many examples where the polyhedral analysis improves on bound propagation, for example the program in Figure 1 of the introduction. Improvement can also be seen in the program `alpha` (see Table 1). Improvements over bound propagation can occur in any program with more than one inequality containing more than one variable. In bound propagation, individual constraints interact with the domain constraints in the store, but are unable to interact with each other. The power of the polyhedral analysis comes from allowing this interaction between constraints in order to achieve better propagation.

It can be seen that the polyhedral method subsumes bound propagation for linear constraints, when both are applied as static analyses. This follows since bound propagation can be viewed as performing Fourier-Motzkin variable elimination on a subset of the inequalities comprising the problem: a subset containing only the bounds from the store and a single inequality with more than one variable. Therefore, as extra information can only lead to tighter bounds, variables will be bounded at least as tightly after Fourier-Motzkin variable elimination for the full problem.

6 Example: Calculating Factorials

This section works through a more complicated example. Performing the analysis automatically on arbitrary (recursive) programs requires machinery which includes, among other things: convex hulls, projection, and widening. These operations are illustrated by the example in this section. The example program calculates factorials. The objective again is to infer bounds on the variables. Usually this reduces searching, but in this case it simply tightens one of the constraints – the point of the example being illustrative. The program (in SICStus syntax) is as follows:

```
:- use_module(library(clpfd)).
fac(0, 1).
fac(N, NewF):-
          N#>=0, NewF#>=0,
          NewF#=N*F,
          M #= N-1,
          fac(M, F).
```

The clause `fac(0,1).` is the first considered. The arguments are described by the polyhedron $P_1 = \{(x, y)|x = 0, y = 1\}$. Next, the second clause is considered. The problem here is to compute a two dimensional polyhedron that describes the coordinate space (N, NewF). First observe that `fac(M, F)` can be described by the polyhedron $\{(\mathtt{N}, \mathtt{NewF}, \mathtt{M}, \mathtt{F})|\mathtt{M} = 0, \mathtt{F} = 1\}$. Note too, that the constraints `M #= N - 1`, `N#>=0`, `NewF#>=0` are represented by the polyhedron $\{(\mathtt{N}, \mathtt{NewF}, \mathtt{M}, \mathtt{F})|\mathtt{M} = \mathtt{N} - 1, \mathtt{N} \geq 0, \mathtt{NewF} \geq 0\}$. The intersection of these two polyhedra, $\{(\mathtt{N}, \mathtt{NewF}, \mathtt{M}, \mathtt{F})|\mathtt{M} = 0, \mathtt{F} = 1, \mathtt{M} = \mathtt{N} - 1, \mathtt{N} \geq 0, \mathtt{NewF} \geq 0\}$, represents the conjunction of the four constraints. The non-linear constraint `NewF#=N*F` cannot, by itself, be accurately represented by a polyhedron. Note, however, that the polyhedron $\{(\mathtt{N}, \mathtt{NewF}, \mathtt{M}, \mathtt{F})|\mathtt{NewF} = \mathtt{N}, \mathtt{M} = 0, \mathtt{F} = 1, \mathtt{M} = \mathtt{N} - 1, \mathtt{N} \geq 0, \mathtt{NewF} \geq 0\}$ accurately describes all the constraints. Projecting the four dimensional polyhedron onto the coordinate space $(\mathtt{N}, \mathtt{NewF})$ gives the polyhedron $\{(\mathtt{N}, \mathtt{NewF})|\mathtt{NewF} = \mathtt{N}, 0 = \mathtt{N} - 1\}$, equivalently $P_2' = \{(x, y)|x = 1, y = 1\}$.

To avoid representing disjunctive information, the solution set $P_1 \cup P_2'$ is over approximated by its convex hull, $P_2'' = \{(x, y)|0 \leq x \leq 1, y = 1\}$. The bound information extracted from the convex hull by projection is exactly the same as that extracted from the union of the original pair of polyhedra by projection. The convex hull gives the second iterate. Continuing in this fashion will give a sequence of increasing polyhedra which does not stabilise. A fixpoint acceleration technique, widening, is therefore used to enforce convergence (albeit at the expense of precision). The widening essentially finds stable bounds on the sequence of polyhedra. P_1 is widened with P_2'' to give the polyhedron $P_2 = \{(x, y)|0 \leq x, y = 1\}$. $P_2 \neq P_1$, and so the fixpoint stability check fails and thus the next iteration is calculated. This results in the polyhedra $P_3' = \{(x, y)|x \geq 1, y \geq 1\}$, $P_3'' = \{(x, y)|x \geq 0, y \geq 1\}$ and $P_3 = \{(x, y)|x \geq 0, y \geq 1\}$. $P_2 \neq P_3$ and stability has still not been reached. However, $P_3 = P_4$, and the fixpoint is found. Projecting P_3 onto the first and second arguments gives the bounds $x \geq 0, y \geq 1$.

Specialising the program by adding these bounds results in the following:

```
:- use_module(library(clpfd)).
fac(0, 1):-
          0#>=0, 1#>=1.
fac(N, NewF):-
```

```
N#>=0, NewF#>=1,
NewF#=N*F,
M #= N-1,
fac(M, F).
```

The redundant constraints in the first clause can be removed. The second clause has one of its domain constraint trivially tightened. The specialisation will always preserve the set of computed answer substitutions.

7 Implementation and Experimental Results

The analysis has been implemented in SICStus Prolog 3.8. The analyser uses rational constraints rather than real constraints as problematic rounding errors occur with the CLP($\mathcal{R}$) package. The `call_residue` built-in that comes as part of the SICStus CLP($\mathcal{Q}$) package is used for projection in this implementation. Other parts of the analyser, such as the convex hull machinery, are taken from [3]. The analyser uses a semi-naïve iteration strategy.

The prototype analyser was tested on a selection of programs from the benchmarks suite that comes with the SICStus release of the CLP($\mathcal{FD}$) package. The programs were chosen for their compatibility with the parser: those programs which passed through the prototype front-end (abstractor) without giving error messages were used.

Bound propagation is applied by the finite domain solver at runtime. To demonstrate that the polyhedral analysis is doing more than shifting some of the work done by bound propagation from runtime to compiletime, experiments were also carried out with bound propagation applied as a compiletime analysis and program transformation.

The programs `alpha`, `crypta`, `donald` and `smm` are all cryptoarithmetic problems. Letters are assigned digits or numbers and equations involving these letters are given. The solution is an assignment of numbers/digits to letters so that the equations are satisfied. The programs `eq10` and `eq20` find solutions to sets of linear equations in seven variables. The program `fac` calculated factorials (in this case 10!). The program `magic` finds magic squares (up to equivalence). The program `five` is a version of the zebra problem, where five lists of five elements are assigned the numbers one to five so that certain relational properties hold. The program `pythagor` calculates Pythagorean triples (in this case with individual values up to one thousand).

The results of the analysis can be seen in Table 1. Vars is the sum of the arities of the predicates that occur in the program; T. Vars is the number of these argument positions tightened by the analysis; Time is the runtime of the original program (in milliseconds); T. Time is the runtime of the specialised program (in milliseconds); Bound Prop. is the runtime of the program when specialised by the values obtained by bound propagation (in milliseconds); Fixpoint is the time taken to calculate the fixpoint (in milliseconds); Fix and Proj. is the runtime of the analysis including the final projection stage (in milliseconds). Note that all times are averages taken over one hundred runs. The experiments were conducted using a PC with a 366MHz Pentium processor and 128Mb of RAM, running Red Hat Linux 6.1.

All but one of the example programs have at least one predicate position tightened by the analysis, indicating that the analysis can be widely applied. No specialised pro-

Program	Vars	T. Vars	Time	T. Time	Bound Prop.	Fixpoint	Fix. and Proj.
`alpha`	26	25	2390	4	2390	280	2100
`crypta`	10	3	6	6	6	460	59430
`donald`	10	3	47	47	47	80	490
`eq10`	7	7	13	0.7	13	100	110
`eq20`	7	7	20	0.2	19	130	140
`fac`^{10}	2	2	0.8	1.0	0.8	260	260
`five`	25	3	1.8	1.8	1.8	100	190
`magic`	9	9	6.5	3.2	6.4	100	230
`smm`	8	3	0.9	0.9	0.9	60	340
`pythagor`^{1000}	3	0	155	154	154	180	190

Table 1. Test Results

gram runs slower than before. After specialisation, the programs `alpha`, `eq10`, `eq20` and `magic` run significantly quicker than both the original programs and the programs specialised by adding the results of bound propagation. This indicates that the analysis can significantly prune the search space. The fixpoint analysis times are reasonable considering that the analyser is a prototype in an early stage of development. In particular, the iteration technique can be improved. Amongst the more expensive fixpoint times are those for `fac` and `pythagor`. These programs are recursive, and although the analyser has been designed to deal with all programs, it is expected that most finite domain programs are not recursive. Notice that the most significant factor in analysing many of the programs is the cost of the final projection stage. The analyser currently uses the projection technique that comes with SICStus. It is to be expected that the use of a projection technique tailored to the specific task of projecting onto a single variable would give significantly improved performance.

8 Related Work

The use of convex polyhedra to describe the constraints in constraint logic programs over the reals has been outlined in [14]. The paper does not describe an implementation and does not directly address the analysis and specialisation of finite domain programs.

The analysis in this paper has its foundations in classic work on polyhedral approximation [5], [6]. Polyhedral approximation has been applied in areas as diverse as: argument size analysis [3]; compiletime array bounds analysis [6]; termination of deductive databases [21]; off-line partial deduction [15]; parallelisation of imperative languages [19]; control generation for logic programs [16]; memory management of symbolic languages based on cdr-coding of lists [8]. The work in this paper directly builds on the work of Benoy and King ([3]) to show how a finite domain program specialiser can be built with off-the-shelf linear constraint solving machinery.

Static analysis of finite domain constraint logic programs is not a new idea. Bagnara [2] proposes an interval analysis for refining domain constraints. The critical observation in this paper is that a finite domain solver will usually perform constraint propagation at runtime, for example through indexical based propagation. The static analysis presented in this paper is designed to complement a runtime constraint analysis: polyhedra capture deep inter-variable relationships which cannot always be traced in bound propagation.

The current work could be viewed as a compiletime approach to the collaboration of constraint solvers. There are many recent papers on collaboration of constraint solvers, such as [17], [20]. Different kinds of constraint solvers will propagate information in different ways, and mixing technologies often gives the best framework for solving a problem. Using a variety of solvers can give propagation that cannot be achieved in a single solver. The approach taken here is attractive because it uses off the shelf technologies and combines their use, but this has the drawback that the propagation is not as intelligent as it might be.

Another compilation technique based on projection arises in providing predictable time-critical user interfaces, [7]. There, however, the objective is to remove runtime constraint solving altogether.

9 Conclusions and Future Work

Analysis of finite domain constraint logic programs using polyhedra promises to be a powerful compiletime technique for reducing the search space of finite domain constraint logic programs. This analysis can extract more information than bound propagation alone. By using program specialisation, other methods of domain reduction can still be applied at runtime. The analysis is safe in two senses: the specialised program is never incorrect; it never runs more slowly that the original. The analysis can be implemented straightforwardly using a rational constraint solver.

The results show that the analysis will tighten the domains of many of the variables in programs – indeed, the analysis completely solves the problems in `eq10` and `eq20`. The timing values in the results table, in particular those for the program `alpha` (where the analysis time plus the tightened time is less than the original time), indicate that polyhedral analysis can give a significant speed up. As a compiletime technique, some extra cost is not prohibitive, however, it is expected that further development will lead to a significant speedup of analysis. The analysis can therefore be considered practical. However, the analysis is not as powerful when data is input at runtime: clearly, in this situation no compiletime specialisation procedure will be effective. The programs with which the analyser has been used have all had the data built into the programs. A wider study of finite domain programs is needed before the significance of this drawback can be assessed.

Future work will focus on developing the analyser. Beyond improving the convex hull and projection calculations, there are several areas where work is in progress. The use of widening could be delayed to improve precision and non-linear constraints could be better approximated. The analyser will also be extended to support other finite domain solvers. It would be an interesting to investigate whether or not it is practical to exploit the extra propagation gained by reanalysing the specialised programs.

Acknowledgements The work of both authors is supported by EPSRC grant number GR/MO8769. The authors would like to thank Florence Benoy, Pat Hill, Jon Martin and Barbara Smith for their helpful comments and suggestions.

References

1. K. R. Apt. A Proof Theoretic View of Constraint Programming. *Fundamenta Informaticae*, 33:1–27, 1998.
2. R. Bagnara. *Data-flow Analysis for Constraint Logic-based Languages*. PhD thesis, Università di Pisa, 1997. TD-1/97.
3. F. Benoy and A. King. Inferring Argument Size Relationships with CLP($\mathcal{R}$). In J. Gallagher, editor, *Logic Program Synthesis and Transformation*, volume 1207 of *Lecture Notes in Computer Science*, pages 204–224. Springer, 1996.
4. A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics Approach: Theory and Applications. *Journal of Logic Programming*, 19-20:149–197, 1994.
5. P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. Technical Report LIENS-92-16, Laboratoire d'Informatique de l'Ecole Normal Superiéure, 1992.
6. P. Cousot and N. Halbwachs. Automatic Discovery of Restraints among Variables of a Program. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 84–97, 1978.
7. H. Harvey, P. J. Stuckey, and A. Borning. Compiling Constraint Solving Using Projection. In *Proceedings of Principles and Practice of Constraint Programming*, volume 1330 of *Lecture Notes in Computer Science*, pages 491–505. Springer, 1997.
8. R. N. Horspool. Analyzing List Usage in Prolog Code. University of Victoria, 1990.
9. J. M. Howe and A. King. A Semantic Basis for Specialising Domain Constraints. Technical Report 21-99, University of Kent, 1999.
10. T. Huynh, C. Lassez, and J.-L. Lassez. Practical Issues on the Projection of Polyhedral Sets. *Annals of Mathematics and Artificial Intelligence*, 6:295–316, 1992.
11. J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proceedings of the Symposium on Principles of Programming Languages*, pages 111–119. ACM Press, 1987.
12. J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19-20:503–582, 1994.
13. K. Marriot and P. J. Stuckey. *Programming With Constraints*. MIT Press, Cambridge, MA., 1998.
14. K. Marriott and P. J. Stuckey. The 3 R's of Optimizing Constraint Logic Programs: Refinement, Removal and Reordering. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, pages 334–344. ACM Press, 1993.
15. J. C. Martin. *Judgement Day: Terminating Logic Programs*. PhD thesis, University of Southampton, 1999.
16. J. C. Martin and A. King. Generating Efficient, Terminating Logic Programs. In *Proceedings of the Seventh International Joint Conference on Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 273–284, Lille, France, 1997. Springer.
17. E. Monfroy. An Environment for Designing/Executing Constraint Solver Collaborations. *Electronic Notes in Theoretical Computer Science*, 16(1), 1998.
18. K. G. Murty. *Linear Programming*. Wiley, 1983.
19. W. Pugh. The Omega Test: a Fast and Practical Integer Programming Algorithm for Dependency Analysis. *Communications of the ACM*, pages 102–114, August 1992.

20. R. Rodošek and M. Wallace. A Generic Model and Hybrid Algorithm for Hoist Scheduling Problems. In *Proceedings of the 4th International Conference on Principals and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*, pages 385–399. Springer, 1998.
21. A. van Gelder. Deriving Constraints Amongst Argument Sizes in Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 3(2-4), 1991.