A Semantic Basis for Specialising Domain Constraints

Jacob M. Howe [1] and Andy King [2]

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK

Abstract

This paper formalises an analysis of finite domain programs and the resultant program transformation. The analysis adds low valency (domain) constraints to clauses in order to reduce search. The technique is outlined with a worked example and then formalised using abstract interpretation. Correctness of the analysis and of the transformation is proved.

1 Introduction

Abstraction interpretation centres on tracing properties of programs using descriptions. In the context of constraint programming, descriptions often capture numeric properties of the store. For example, LSign is useful for checking the satisfiability of linear constraints [11,13]; intervals have been proposed for refining domain constraints of finite domain programs [1]; polyhedra have been applied to optimise CLP($\mathcal{R}$) programs [10]. To obtain finiteness, analyses usually trace information in an approximation of the concrete domain. This paper, however, uses a slightly different tactic. Finite domain constraint programs are reinterpreted as constraint programs over linear equations, and polyhedral abstraction is then applied to propagate information in this domain. This enables information to be inferred which cannot be deduced with an approximation of the concrete domain.

Howe and King argue in [6] that constraint propagation performed at compile-time by an analysis should complement the run-time propagation of the solver. Specifically, they demonstrate that a significant speedup (sometimes of several orders of magnitude) can be obtained by using polyhedra to infer deep inter-variable relationships in finite domain programs which cannot be traced by run-time (interval based) bound propagation. The crucial tactic is to combine the constraints deduced by the

[1] Supported by EPSRC grant GR/MO8769.
[2] Supported, in part, by COTIC working group 23677.

analysis with a program specialisation based on projection. To be precise, finite domain constraints are interpreted as relations over sets of points. These constraints are over-approximated and represented as a 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 speedup follows from reducing the search. Interestingly, the analysis can be interpreted as a compile-time solution to combining constraint solvers [12].

This paper formalises the analysis of [6] in terms of abstract interpretation. Correctness of the analysis and of the associated program transformation is established. The analysis is constructed in terms of operations on polyhedra, for example, calculating the closure of the convex hulls of polyhedra, and also uses fixed-point acceleration techniques, such as widening, to obtain convergence. Correctness is proved with respect to a ground fixpoint semantics for (definite) constraint logic programs [7]. The analysis does not preserve the set of computed answer constraints (but increases it).

Work that is particularly closely related to this paper is an analysis of deductive database queries [9] that uses polyhedral abstractions to propagate constraints. The current paper applies similar abstraction techniques, though the analysis and the transformation differ significantly in detail. One crucial difference in the work presented here is the way that projection is used to constrain individual program variables of finite domain programs with domain constraints. Without this step, the analysis would have little effect.

The structure of the paper is as follows: section 2 works through an example program to illustrate how the analysis works; section 3 gives details of the abstract interpretation; section 4 proves the correctness of the analysis; section 5 describes the program transformation and establishes its correctness; and finally, section 6 concludes.

2 Calculating Factorials

This section works through an example to illustrate the polyhedral analysis. The analysis requires machinery which includes: the closure of convex hulls, projection, and widening. The example program calculates factorials. The objective is to infer bounds on the variables. Usually this reduces searching, but in this case it only improves the termination behaviour of the program. The program (in SICStus syntax) is:

```
:- use_module(library(clpfd)).
fac(0, 1).
fac(N, NewF) :-
       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 constraint `M #= N - 1` is represented by the polyhedron $\{(\mathtt{N}, \mathtt{NewF}, \mathtt{M}, \mathtt{F})|\mathtt{M} = \mathtt{N} - 1\}$. The intersection of these two polyhedra, $\{(\mathtt{N}, \mathtt{NewF}, \mathtt{M}, \mathtt{F})|\mathtt{M} = 0, \mathtt{F} = 1, \mathtt{M} = \mathtt{N}-1\}$, represents the conjunction of the two constraints. The non-linear constraint `NewF#=N*F` cannot, by itself, be accurately represented by a polyhedron. Note, however, that the polyhedron $\{(\mathtt{N}, \mathtt{NewF}, \mathtt{M}, \mathtt{F})|\mathtt{NewF} = \mathtt{N}, \mathtt{M} = 0, \mathtt{F} = 1, \mathtt{M} = \mathtt{N} - 1\}$ accurately describes all the constraints. Projecting the four dimensional polyhedron onto the coordinate space $(\mathtt{N}, \mathtt{NewF})$ gives the polyhedron $\{(\mathtt{N}, \mathtt{NewF})|\mathtt{NewF} = \mathtt{N}, 0 = \mathtt{N}-1\}$, or 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 the closure of the convex hull, $P_2'' = \{(x, y)|0 \leq x \leq 1, y = 1\}$. (Note that the bound information extracted by projection from the convex hull is exactly the same as that extracted by projection from the union of the original pair of polyhedra.) P_2'' is 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 force convergence, albeit at the expense of some 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. Continuing as before 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 low valency constraints results in:

```
:- 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. Note that the specialised program has improved termination behaviour. For example, the queries `fac(-1,_)` and `fac(_, 5)` fail, whereas previously both led to non-terminating computations. More generally, the experimental work and benchmarking reported in [6] suggests that this technique can often significantly improve performance.

3 Polyhedral Analysis

This section formalises and describes the analysis. Abstract interpretation is used to connect a (concrete) ground semantics for finite domain constraint programs [7,8] to an (abstract) s-semantics [3]. A Galois insertion links the concrete domain (the set of ground interpretations) and the abstract domain (the set of interpretations over constrained unit clauses). Convex hulls are used to obtain a small non-ground interpretation and widening is used to ensure termination.

3.1 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 constant, 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. C_{FD} is the system of finite domain constraints generated from D_{FD} and R_{FD}. 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 $\exists x_1 \ldots \exists x_n.c$. $\overline{\exists}Xc$ (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.

3.2 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} and the set of constraint predicates R_{Lin}. 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 $\forall[p(\overline{\mathbf{x}}) \leftarrow c]_\equiv \in I.\exists[p(\overline{\mathbf{x}}) \leftarrow c']_\equiv \in I'.c \models_{Lin} c'$. $\mathcal{P}(B_{Lin})$ 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 $\sqcup_{i=1}^{\infty}[I_i]_\equiv = [\cup_{i=1}^{\infty} I_i]_\equiv$).

Proposition 1 *$[\cup_{i=1}^{\infty} I_i]_\equiv$ is the least upper bound of $\{[I_i]_\equiv | i \in \mathbb{N}\}$.*

PROOF: Since $I_i \subseteq \cup_{i=1}^{\infty} I_i$, it follows that $I_i \sqsubseteq \cup_{i=1}^{\infty} I_i$ and therefore $[I_i]_\equiv \sqsubseteq [\cup_{i=1}^{\infty} I_i]_\equiv$. Suppose $[I_i]_\equiv \sqsubseteq [J]_\equiv$ for all $i \in \mathbb{N}$ and let $[p(\overline{\mathbf{x}}) \leftarrow c]_\equiv \in \cup_{i=1}^{\infty} I_i$. There must exist $j \in \mathbb{N}$ such that $[p(\overline{\mathbf{x}}) \leftarrow c]_\equiv \in I_j$. Hence there exists $[p(\overline{\mathbf{x}}) \leftarrow c']_\equiv \in J$ such that $c \models_{Lin} c'$. Thus $\cup_{i=1}^{\infty} I_i \sqsubseteq J$. ■

3.3 Concretisation

A concretisation map is defined in terms of a map, $\cdot^+ : C_{Lin} \rightarrow C_{FD}$, that interprets a linear constraint over the rationals as a finite domain constraint as follows:

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

Note that the coefficients of c_{Lin}^+ are in $\mathbb{Z}$. The concretisation map, $\gamma : C_{Lin} \rightarrow C_{FD}$, is simply defined as

$$\gamma(c_{Lin}) = c_{Lin}^+$$

The abstraction map, $\alpha : C_{FD} \rightarrow C_{Lin}$ can be defined in terms of γ as follows:

$$\alpha(c_{FD}) = \wedge\{c_{Lin} | c_{FD} \models_{FD} \gamma(c_{Lin})\}$$

Lemma 1 *α, γ form a Galois insertion.*

PROOF: First observe that α and γ are monotonic. By definition

$$\alpha(\gamma(c_{Lin})) = \wedge\{c'_{Lin} | \gamma(c_{Lin}) \models_{FD} \gamma(c'_{Lin})\} \models_{Lin} c_{Lin}$$

This shows that there is a Galois connection. To prove that there is a Galois insertion, injectivity of γ is demonstrated. Given that $\gamma(c) = \gamma(c')$, assume that $c \neq c'$. Without loss of generality, $c' \not\models_{Lin} c$, hence $c'^+ \not\models_{FD} c^+$. Together with $\gamma(c') = \gamma(c) \models_{FD} c^+$, this gives $\gamma(c') \not\models_{FD} c'^+$. This contradicts the definition of γ, therefore $c = c'$. ■

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

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

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 2 *The concretisation map on interpretations, γ, is monotonic and therefore well-defined.*

PROOF: Suppose $[I]_\equiv \sqsubseteq [I']_\equiv$. Let $p(\bar{\mathbf{t}}) \in \gamma([I]_\equiv)$. Then there exists $[p(\bar{\mathbf{x}}) \leftarrow c]_\equiv \in I$ such that $(\bar{\mathbf{x}} = \bar{\mathbf{t}}) \models_{FD} \gamma(c)$. There exists $[p(\bar{\mathbf{x}}) \leftarrow c']_\equiv \in I'$ such that $c \models_{Lin} c'$. But, $(\bar{\mathbf{x}} = \bar{\mathbf{t}}) \models_{FD} \gamma(c) \models_{FD} \gamma(c')$ and hence $p(\bar{\mathbf{t}}) \in \gamma([I']_\equiv)$. ■

Proposition 3 *α and γ on interpretations form a Galois insertion.*

PROOF: Observe that α is monotonic and, by Proposition 2, γ is also. By the definitions of α and γ on interpretations and Lemma 1,

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

This shows that there is a Galois connection. To prove that there is a Galois insertion, injectivity of γ is demonstrated. Given that $\gamma([I]_\equiv) = \gamma([I']_\equiv)$, assume $[I]_\equiv \neq [I']_\equiv$. Then $\exists [p(\bar{\mathbf{x}}) \leftarrow c]_\equiv \in I$ such that $\forall [p(\bar{\mathbf{x}}) \leftarrow c']_\equiv \in I'.c \not\models_{Lin} c'$. Thus $\gamma(c) \not\models_{FD} \gamma(c')$. By the definition of γ, $(\bar{\mathbf{x}} = \bar{\mathbf{t}}) \models_{FD} \gamma(c) \not\models \gamma(c')$. This contradicts the assumption and therefore $[I]_\equiv = [I']_\equiv$. ■

3.4 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 $\mathcal{F}_{FD}[\![P]\!] = lfp(T_P^g)$.

3.5 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, $T_P^s : \mathcal{P}(B_{Lin})/\!\equiv \rightarrow \mathcal{P}(B_{Lin})/\!\equiv$, defined by $T_P^s([I]_\equiv) = [J]_\equiv$, where

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

T_P^s is continuous, thus $lfp(T_P^s)$ exists. Since $\mathcal{P}(B_{Lin})/\equiv$ is a complete partial order, Kleene iteration [4] 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 $T_P^s \uparrow i+1 = T_P^s(T_P^s \uparrow i)$.

3.6 Space-Efficient Over-Approximation

$T_P^s \uparrow 1$ can contain many unit clauses for the same predicate. Furthermore, $T_P^s \uparrow 2$ will contain as many, if not more, unit clauses. Thus, to make the fixpoint calculation manageable, $T_P^s \uparrow k$ is over-approximated by an interpretation I (that is, $T_P^s \uparrow k \sqsubseteq I$) which contains at most one unit clause for each predicate symbol.

The join for the domain of linear constraints, $\vee : C_{Lin} \times C_{Lin} \rightarrow C_{Lin}$, 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, that is, the smallest enclosing closed convex set. The operator is lifted in stages to an operator on the abstract domain. First it is lifted to the interpretation base, $\vee : B_{Lin}^\perp \times B_{Lin}^\perp \rightarrow B_{Lin}^\perp$, where $B_{Lin}^\perp = B_{Lin} \cup \{\perp\}$, as follows:

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

This in turn defines the unary function, $\vee : \mathcal{P}(B_{Lin})/\!\equiv \rightarrow \mathcal{P}(B_{Lin})/\!\equiv$, 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.

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, a fixpoint acceleration technique, called widening [4], is applied.

A widening, ∇, on a partially ordered set $(L, \sqsubseteq, \sqcup)$ is an operator $\nabla : L \times L \to L$ such that $\forall x, y \in L. x \sqsubseteq x \nabla y$ and $\forall x, y \in L. y \sqsubseteq x \nabla y$ and for all increasing chains $x_0 \sqsubseteq x_1 \sqsubseteq ...$, the increasing chain defined by $y_0 = x_0, ..., y_{i+1} = y_i \nabla x_{i+1}, ...$ stabilises, that is, for some m, $y_{m+1} \sqsubseteq y_m$.

Given a standard widening on polyhedra [2,4,5] (or equivalently, on linear constraints), $\nabla : C_{Lin} \times C_{Lin} \to C_{Lin}$, a widening, $\nabla : B_{Lin}^{\perp} \times B_{Lin}^{\perp} \to B_{Lin}^{\perp}$, (where $B_{Lin}^{\perp} = B_{Lin} \cup \{\perp\}$) on the interpretation base is induced as follows:

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

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

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

Proposition 4 ∇ *on interpretations is a widening.*

4 Correctness of the Polyhedral Analysis

This section gives a proof of the correctness of the analysis. That is, upward iteration of $\vee \circ T_P^s$, with widening, stabilises at an interpretation I with $lfp(T_P^g) \sqsubseteq \gamma(I)$.

Lemma 2 *If* $I_{FD} \subseteq \gamma([I_{Lin}]_{\equiv})$*, then* $T_P^g(I_{FD}) \subseteq \gamma(T_P^s([I_{Lin}]_{\equiv}))$.

PROOF: Suppose $p(\mathbf{\overline{t}}) \in T_P^g(I_{FD})$. To prove the result it needs to be shown that there is $[p(\mathbf{\overline{x}}) \leftarrow c]_{\equiv} \in T_P^s([I_{Lin}]_{\equiv})$ such that $(\mathbf{\overline{x}} = \mathbf{\overline{t}}) \models_{FD} \gamma(c)$. By the hypothesis and the definition of $T_P^g(I_{FD})$, let $p(\mathbf{\overline{t}}_i) \in I_{FD} \subseteq \gamma([I_{Lin}]_{\equiv})$. From the definition of $\gamma([I_{Lin}]_{\equiv})$, there is $[p(\mathbf{\overline{y}}_i) \leftarrow c_i]_{\equiv} \in I_{Lin}$ such that $(\mathbf{\overline{y}}_i = \mathbf{\overline{t}}_i) \models_{FD} \gamma(c_i)$

$$\begin{array}{ll}
\Rightarrow & (\mathbf{\overline{x}}_i = \mathbf{\overline{y}}_i)^+ \wedge (\mathbf{\overline{y}}_i = \mathbf{\overline{t}}_i) \models_{FD} \gamma((\mathbf{\overline{x}}_i = \mathbf{\overline{y}}_i) \wedge c_i) \\
\Rightarrow & \wedge_{i=1}^n ((\mathbf{\overline{x}}_i = \mathbf{\overline{y}}_i)^+ \wedge (\mathbf{\overline{y}}_i = \mathbf{\overline{t}}_i)) \models_{FD} \gamma(\wedge_{i=1}^n ((\mathbf{\overline{x}}_i = \mathbf{\overline{y}}_i) \wedge c_i)) \\
\Rightarrow & c' \wedge \wedge_{i=1}^n ((\mathbf{\overline{x}}_i = \mathbf{\overline{y}}_i)^+ \wedge (\mathbf{\overline{y}}_i = \mathbf{\overline{t}}_i)) \models_{FD} \gamma(c' \wedge_{FL} \wedge_{i=1}^n ((\mathbf{\overline{x}}_i = \mathbf{\overline{y}}_i) \wedge c_i)) \\
\Rightarrow & \overline{\exists} var(\mathbf{\overline{x}})(c' \wedge \wedge_{i=1}^n ((\mathbf{\overline{x}}_i = \mathbf{\overline{y}}_i)^+ \wedge (\mathbf{\overline{y}}_i = \mathbf{\overline{t}}_i))) \models_{FD} \gamma(\overline{\exists} var(\mathbf{\overline{x}})(c' \wedge_{FL} \wedge_{i=1}^n ((\mathbf{\overline{x}}_i = \mathbf{\overline{y}}_i) \wedge c_i)))
\end{array}$$

By the definition of T_P^g, $(\overline{\mathbf{x}} = \overline{\mathbf{t}}) \models_{FD} \overline{\exists} var(\overline{\mathbf{x}})(c' \wedge \wedge_{i=1}^n (\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i)) \models_{FD} \overline{\exists} var(\overline{\mathbf{x}})(c' \wedge \wedge_{i=1}^n ((\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i)^+ \wedge (\overline{\mathbf{y}}_i = \overline{\mathbf{t}}_i)))$ and therefore

$$(\overline{\mathbf{x}} = \overline{\mathbf{t}}) \models_{FD} \gamma(\overline{\exists} var(\overline{\mathbf{x}})(c' \wedge_{FL} \wedge_{i=1}^n (\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i \wedge c_i)))$$

Put $c = \overline{\exists} var(\overline{\mathbf{x}})(c' \wedge_{FL} \wedge_{i=1}^n (\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i \wedge c_i))$, then $(\overline{\mathbf{x}} = \overline{\mathbf{t}}) \models_{FD} \gamma(c)$ and $[p(\overline{\mathbf{x}} \leftarrow c]_\equiv \in T_P^s([I_{Lin}]_\equiv)$, therefore $p(\overline{\mathbf{t}}) \in \gamma(T_P^s([I_{Lin}]_\equiv))$. ■

Lemma 3 *If $I_{FD} \subseteq \gamma(I_{Lin})$, then $T_P^g(I_{FD}) \subseteq \gamma(\vee \circ T_P^s(I_{Lin}))$*

PROOF: $\gamma(T_P^s(I_{Lin})) \subseteq \gamma(\vee \circ T_P^s(I_{Lin}))$ follows from the definition of the convex hull operator and the monotonicity of γ. The result then follows from Lemma 2. ■

Lemma 4 $[I]_\equiv \sqsubseteq [I']_\equiv$ *iff* $\gamma([I]_\equiv) \subseteq \gamma([I']_\equiv)$.

PROOF: Case 1. $\Leftarrow$. Suppose $\gamma([I]_\equiv) \subseteq \gamma([I']_\equiv)$. Then there exists $[p(\overline{\mathbf{x}}) \leftarrow c]_\equiv \in I$ such that for all $[p(\overline{\mathbf{x}}) \leftarrow c']_\equiv \in I'$, $c \not\models_{Lin} c'$. This contradicts the hypothesis, and the results follows.

Case 2. $\Rightarrow$. This follows from Proposition 2. ■

Proposition 5 below is asserted and proved as Proposition 13 in [4].

Proposition 5 *If $(L, \preceq, \curlyvee)$ is a partially ordered set, $F : L \to L$ is continuous, $\bot \in L$ is such that $\bot \preceq F(\bot)$, $\curlyvee_{n\in\mathbb{N}} F^n(\bot)$ exists, $\hat{L}$ is a set, $\gamma : \hat{L} \to L$, $\hat{\preceq}$ is the preorder defined by $x \hat{\preceq} y$ iff $\gamma(x) \preceq \gamma(y)$, $\hat{\bot} \in \hat{L}$ is such that $\bot \preceq \gamma(\hat{\bot})$, $\hat{F} : \hat{L} \to \hat{L}$ is monotonic, $F \circ \gamma \preceq \gamma \circ \hat{F}$ and $\nabla : \hat{L} \times \hat{L} \to \hat{L}$ is a widening, then the upward iteration sequence with widening is ultimately stationary with limit $\hat{A}$ such that $lfp(F) \preceq \gamma(\hat{A})$ and $\hat{F}(\hat{A}) \hat{\preceq} \hat{A}$.*

Corollary 1 *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 $lfp(T_P^g) \subseteq \gamma(I)$.*

PROOF: Using Lemma 3 and Lemma 4 above, Proposition 5 can be applied, giving the result. ■

5 Program Transformation and its Correctness

Once an upper approximation to $\mathcal{F}_{FD}[\![P]\!]$ is computed, it can be used to transform the program. The following theorem details the transformation and also asserts safety.

An auxiliary (partial) map, $\cdot^t : C_{Lin} \to C_{Lin}$, 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*

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

PROOF: The proof proceeds by showing, by induction on n, that $T_P^g \uparrow n = T_{P'}^g \uparrow n$.

Base Case: $T_P^g \uparrow 0 = \phi = T_{P'}^g \uparrow 0$.

Inductive case: Suppose that $T_P^g \uparrow k = T_{P'}^g \uparrow k$,

(1) To show $T_{P'}^g \uparrow (k+1) \subseteq T_P^g \uparrow (k+1)$. Let $p(\overline{\mathbf{t}}) \in T_{P'}^g \uparrow (k+1) = T_{P'}^g(T_{P'}^g \uparrow k)$. Thus there exists $w' \in P'$ such that $w' = p(\overline{\mathbf{x}}) \leftarrow (c \wedge c'), p_1(\overline{\mathbf{x}}_1), ..., p_n(\overline{\mathbf{x}}_n)$ (where $w = p(\overline{\mathbf{x}}) \leftarrow c, p_1(\overline{\mathbf{x}}_1), ..., p_n(\overline{\mathbf{x}}_n)$), $p_i(\overline{\mathbf{t}}_i) \in T_{P'}^g \uparrow k$ and $(\overline{\mathbf{x}} = \overline{\mathbf{t}}) \models_{FD} \overline{\exists} var(\overline{\mathbf{x}})(\wedge_{i=1}^n(\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i) \wedge (c \wedge c'))$. Therefore, $(\overline{\mathbf{x}} = \overline{\mathbf{t}}) \models_{FD} \overline{\exists} var(\overline{\mathbf{x}})(\wedge_{i=1}^n(\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i) \wedge c)$ and $p(\overline{\mathbf{t}}_i) \in T_{P'}^g \uparrow k \subseteq T_P^g \uparrow k$. Therefore $p(\overline{\mathbf{t}}) \in T_P^{k+1}$.

(2) To show $T_P^g \uparrow (k+1) \subseteq T_{P'}^g \uparrow (k+1)$. Let $p(\overline{\mathbf{t}}) \in T_P^g \uparrow (k+1) = T_P^g(T_P^g \uparrow k)$. Then there exists $w \in P$ such that $w = p(\overline{\mathbf{x}}) \leftarrow c, p_1(\overline{\mathbf{x}}_1), ..., p_n(\overline{\mathbf{x}}_n)$ and $p_i(\overline{\mathbf{t}}_i) \in T_P^g \uparrow k$, $(\overline{\mathbf{x}} = \overline{\mathbf{t}}) \models_{FD} \overline{\exists} var(\overline{\mathbf{x}})(\wedge_{i=1}^n(\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i) \wedge c)$. Since $T_P^g \uparrow k \subseteq lfp(T_P^g) \subseteq \gamma([I]_\equiv)$ and since there is $[p(\overline{\mathbf{y}}_i) \leftarrow c_i]_\equiv \in I$ such that $(\overline{\mathbf{y}}_i = \overline{\mathbf{t}}_i) \models_{FD} \gamma(c_i)$, then

$$\begin{array}{llll}
\Rightarrow & (\overline{\mathbf{y}}_i = \overline{\mathbf{t}}_i) & \models_{FD} & c_i^+ \\
\Rightarrow & (\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i) \wedge (\overline{\mathbf{y}}_i = \overline{\mathbf{t}}_i) & \models_{FD} & ((\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i) \wedge c_i)^+ \\
\Rightarrow & (\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i) & \models_{FD} & (\exists var(\overline{\mathbf{y}}_i)((\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i) \wedge c_i))^+ \\
\Rightarrow & \wedge_{i=1}^n(\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i) & \models_{FD} & (\wedge_{i=1}^n \exists var(\overline{\mathbf{y}}_i)((\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i) \wedge c_i))^+ \\
\Rightarrow & \wedge_{i=1}^n(\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i) & \models_{FD} & (\overline{\exists} y(\wedge_{i=1}^n \exists var(\overline{\mathbf{y}}_i)((\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i) \wedge c_i)))^+ \\
\Rightarrow & \wedge_{i=1}^n(\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i) & \models_{FD} & ((\overline{\exists} y(\wedge_{i=1}^n((\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i) \wedge c_i)))^t)^+ \\
\Rightarrow & \wedge_{i=1}^n(\overline{\mathbf{x}}_i = \overline{\mathbf{t}}_i) & \models_{FD} & \wedge_{y \in var(w)}((\overline{\exists} y(\wedge_{i=1}^n((\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i) \wedge c_i)))^t)^+
\end{array}$$

Putting $c' = c \wedge \wedge_{y \in var(w)}((\overline{\exists} y(\wedge_{i=1}^n((\overline{\mathbf{x}}_i = \overline{\mathbf{y}}_i) \wedge c_i)))^t)^+$ and $w' = p(\overline{\mathbf{x}}) \leftarrow c', p_1(\overline{\mathbf{x}}_1), ..., p_n(\overline{\mathbf{x}}_n)$ it follows that $p(\overline{\mathbf{t}}) \in T_{P'}^g \uparrow (k+1)$.

Thus $T_P^g \uparrow k+1 = T_{P'}^g \uparrow k+1$ and so, by induction, $lfp(T_P^g) = lfp(T_{P'}^g)$. ■

6 Conclusions

Analysis of finite domain constraint logic programs using polyhedra promises to be a powerful compile-time technique for reducing the search space of, and thereby speeding up, finite domain constraint logic programs. By using program specialisation, other methods of domain reduction can still be applied at run-time. This paper has formally established the correctness of a polyhedral analysis and of the associated transformation which adds low valency constraints to the program. The technique is safe in two senses: the specialised program is never incorrect; it never runs (significantly) more slowly that the original.

Acknowledgements

The authors would like to thank Florence Benoy, Pat Hill, Jon Martin and Barbara Smith for their helpful comments and suggestions.

References

[1] R. Bagnara. *Data-flow Analysis for Constraint Logic-based Languages*. PhD thesis, Università di Pisa, 1997. TD-1/97.

[2] 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 *LNCS*, pages 204–224. Springer-Verlag, 1996.

[3] 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.

[4] 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.

[5] 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.

[6] J. M. Howe and A. King. Specialising Finite Domain Programs Using Polyhedra. Submitted to Logic Program Synthesis and Transformation, 1999.

[7] 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.

[8] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19-20:503–582, 1994.

[9] D. B. Kemp and P. J. Stuckey. Analysis Based Constraint Query Optimization. In *Proceedings of the 1993 International Conference on Logic Programming*, pages 666–682. MIT Press, 1993. Long version available from `http://www.cs.mu.oz.au/~kemp/`.

[10] 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.

[11] K. Marriott and P. J. Stuckey. Approximating Interaction Between Linear Arithmetic Constraints. In *Proceedings of the International Symposium on Logic Programming*. MIT Press, 1994.

[12] E. Monfroy. An Environment for Designing/Executing Constraint Solver Collaborations. *Electronic Notes in Theoretical Computer Science*, 16(1), 1998.

[13] V. Ramachandran and P. Van Hentenryck. LSign Reordered. In *Proceedings of the Second International Symposium on Static Analysis*, volume 983 of *LNCS*, pages 330–347. Springer-Verlag, 1995.