# Inferring Argument Size Relationships with CLP($\mathcal{R}$)

Florence Benoy and Andy King

Computing Laboratory,
University of Kent at Canterbury, CT2 7NF, UK.
{p.m.benoy, a.m.king}@ukc.ac.uk

**Abstract.** Argument size relationships are useful in termination analysis which, in turn, is important in program synthesis and goal-replacement transformations. We show how a precise analysis for inter-argument size relationships, formulated in terms of abstract interpretation, can be implemented straightforwardly in a language with constraint support like CLP($\mathcal{R}$) or SICStus version 3. The analysis is based on polyhedral approximations and uses a simple relaxation technique to calculate least upper bounds and a delay method to improve the precision of widening. To the best of our knowledge, and despite its simplicity, the analysis derives relationships to an accuracy that is either comparable or better than any existing technique.

## 1 Introduction

Termination analysis is important in program synthesis, goal-replacement transformations and is also likely to be useful in off-line partial deduction. Termination analysis is usually necessary in synthesis since synthesis often only guarantees semantic or model-theoretic correctness. Termination analysis is often necessary in transformation because termination usually must be preserved wrt. the initial program and transformed goals. Termination is typically proved by showing that a well-founded ordering exists between a goal and its sub-goals. In the case of the `Qs/2` program, termination is (basically) asserted by showing that the recursive `Qs(l, sl)` and `Qs(g, sg)` goals operate on lists that are strictly smaller than `[x|xs]`. (For the definition of the `Ap/2` and `Pt/4` predicates see appendix A.) For programs like `Qs/2` which are not structurally recursive this, in turn, requires the derivation of inter-argument relationships. In the case `Qs/2`, for instance, it is necessary to infer that both `l` and `g` in the recursive calls are smaller than `[x|xs]`. This can only be inferred by deducing an inter-argument relation for `Pt/4`, that is, that neither the third nor fourth argument are larger than the second argument.

```
Qs([], []).                     Qs^A(0, 0).
Qs([x|xs], s) <-                Qs^A(1+xs, s) <-
   Pt(x, xs, l, g) &               Pt^A(_, xs, l, g) &
   Qs(l, sl) & Qs(g, sg) &         Qs^A(l, sl) & Qs^A(g, sg) &
   Ap(sl, [x|sg], s).              Ap^A(sl, 1+sg, s).
```

Note that Gödel notation is used throughout: variables are denoted by identifiers beginning with a lower case letter and constants by identifiers beginning with an upper case letter.

Once an appropriate measure of term size (norm) like list length, is deduced [12, 23], the problem of inferring argument relationships is essentially reduced to that of inferring invariants of a CLP($\mathcal{R}$) program [14]. `Qs`$^{\mathcal{A}}$`/2` for example, an abstraction of `Qs/2`, is a form of abstract program [14] that is obtained by a syntactic transformation in which each term in the first program is replaced by its size wrt. list length. An analysis for inferring invariants between the variables of the second program [9, 20] can then be re-interpreted as an analysis for deducing the size invariants (inter-argument relationships) of the first program [8, 31]. For example, one invariant in the second program is that the third and fourth arguments of `Pt/4` sum to the second argument. Thus, in the first program, the sum of the lengths of the third and fourth arguments of `Pt/4` must be coincident with the length of the second argument.

In broad terms, analyses for inferring invariants have either been built around affine sub-spaces [14, 20, 31], or in terms of (closed) polyhedral convex sets [8, 18, 30]. (The difference equation approach [11] to inferring inter-argument relationships, although potentially useful, requires computer algebra machinery to manipulate and solve the difference equations and therefore is probably too complicated for most partial deduction systems.) In the affine approach, invariants are represented by affine subspaces, basically points, lines or hyper-planes in $\mathbb{R}^n$, which can be represented and manipulated using matrices. The affine approach is attractive because, although is cannot express inequalities between variables, the approximation is Noetherian and therefore the termination of the analysis is not an issue.

The convex set approach characterises argument relationships as sets of conjoined linear inequalities [18, 30]. To be precise, linear inequalities represent a collection of closed half-spaces the intersection of which, defines a polyhedral convex set. In [30], a suite of transformations are defined, formulated in terms of matrices, for mechanising the derivation of argument relationships. This approach is promising because inequalities are more expressive than equalities since every affine sub-space is polyhedral.

```
Sp([], [], []).                    SpA(0, 0, 0).
Sp([x|xs], [x|os], es) <-          SpA(1+xs, 1+os, es) <-
   Sp(xs, es, os).                    SpA(xs, es, os).
```

To illustrate the expressiveness of inequalities, consider the `Sp/3` predicate [27] of merge sort in which the elements at the odd and even positions in a list are separated into two lists. The query `<- Sp([A, B, C], o, e).` will succeed with the answer {`e = [B], o = [A, C]`}. Polyhedral approximations are expressiveness enough to even describe the invariants of `Sp/3`, that is,

$$\left\{ \langle x, y, z\rangle \in \mathbb{R}^3 \;\middle|\; \begin{array}{l} z = (-y) + x \ \wedge \ y - x \leq 0 \ \wedge \\ z \leq y \leq z + 1 \ \wedge \ (-y) \leq 0 \end{array} \right\}$$

The polyhedral work of [30] is incomplete, however, because the iterative process required to compute argument relationships for recursive predicates may not converge in finitely many steps. Cousot and Cousot explain, however, how to rectify this problem with widening [8]. Widening essentially trades precision for finiteness by weakening inequality constraints to obtain stability of the iterates.

Our contribution is to show how a precise analysis based on polyhedra (rather than affine sub-spaces [14]) can be implemented straightforwardly in a language with constraint support like CLP($\mathcal{R}$) or SICStus version 3. In fact the initial prototype is less than 200 clauses and took just two person weeks to code and debug. Specifically, we adopt a relaxation technique used in disjunctive constraint programming [10] to compute convex hulls. With the use of the solver and projection machinery of CLP($\mathcal{R}$) and the clp(Q,R) libraries of SICStus, it has not been necessary to manipulate matrices, like [20, 30, 31]; or frames, like [9, 15]; or implement the Chernikova conversion mechanism, like [32].

Convergence of the iterates is enforced by widening and it is our observation that precision can be improved by delaying widening for a few (typically one or two) iterations. This simple approach seems to achieve comparable or better results to the more sophisticated widening of [8], without loss of precision. The principal advantage is in the simplicity in the implementation.

The applications of inferring argument relationships extend well beyond partial deduction. Argument relationships are useful for planning the evaluation of queries in deductive databases [29], optimising database queries [21], and play an important role in time-complexity analysis [11]. Horspool [18] proposed the use of argument relationships for improving the memory management of cdr-coded lists. Also, in Reform compilation [25], where bounded iteration (the `for` loop) is used to implement recursion to avoid the overheads of run-time unfolding, argument relationships can extend the scope for parallelisation by recognising predicates that are defined by structural recursion. Intuitively, this means that the compiler can deduce the recursion bound by just looking at the input arguments [26].

The exposition is structured as follows. Section 2 outlines the analysis with a worked example. Section 3 present some theory and notation to aid the presentation. Sections 4 and 5 cover the convex hull calculation and widening operation. Section 6 outlines the implementation and finally Sections 7 and 8 present the related and future work. The table in appendix A summaries some interesting analysis results obtained by our analyser.

## 2 Worked Example

Consider an argument size analysis for the predicate `Ap/3`. As with `Qs/2`, analysis is performed on an abstract program, here denoted $\texttt{Ap}^{\mathcal{A}}\texttt{/3}$. The arguments of each predicate in the abstract program represent the sizes of the arguments of the corresponding predicate in the concrete program. Therefore the relationships that hold between arguments of $\texttt{Ap}^{\mathcal{A}}\texttt{/3}$ exist as inter-argument size relationships for `Ap/3` the concrete program.

```
Ap([], s, s).                    ApA(0, s, s).
Ap([x|xs], s, [x|t]) <-          ApA(1 + r, s, 1 + t) <-
   Ap(xs, s, t).                    ApA(r, s, t).
```

Analysis iterates to a fixpoint that characterises the inter-argument relationships. We denote the $i^{th}$ iterate by $I_i$. Each iteration in the fixpoint calculation takes an $I_i$ as input and generates an $I_{i+1}$ as output. $I_0$, the bottom element, is $\emptyset$. Generally, to compute $I_{i+1}$, the body atoms of each clause of the program are unified with the atoms in $I_i$. Since $I_0$ is empty, however, $I_1$ will abstract only those relationships embodied in the unit clause of $\texttt{Ap}^{\mathcal{A}}\texttt{/3}$, that is,

$$I_1 = \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid r \leq 0 \wedge -r \leq 0 \wedge s - t \leq 0 \wedge t - s \leq 0\}$$

Note that $I_1$ is expressed in terms of a set of inequalities. Thereafter, at each iteration, there will be a set of inequalities that describe the inter-argument relationships for each predicate. The number of atoms can grow at each iteration and therefore, to keep the size of the iterate small, the sets of inequalities for each predicate are collected and approximated by an over-estimate, a convex hull. The convex hull can itself be expressed as a single set of inequalities so that it is necessary only to maintain one set of inequalities for each predicate in the program. The convex hull derives a succinct expression of the disjunction of spaces with a minimal loss of information. The convex hull operation denoted $\overline{\cup}$, is used to compute $I_2$ and the ensuing iterates.

$$\begin{aligned} I_2 &= \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid r = 0 \wedge s = t\} \overline{\cup} \{\langle 1 + r, s, 1 + t\rangle \in \mathbb{R}^3 \mid r = 0 \wedge s = t\} \\ &= \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid r = 0 \wedge s = t\} \overline{\cup} \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid r = 1 \wedge s = t - 1\} \\ &= \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid 0 \leq r \wedge r \leq 1 \wedge t = r + s\} \end{aligned}$$

The equalities denote pairs of inequalities for brevity. Although the convex hull operation computes an approximation, useful relationships are still preserved since the convex hull corresponds to the smallest convex space enclosing the spaces represented by the sets of inequalities. Note too, the convex hull calculation effectively generates inter argument relationships, like $t = r + s$, that are common to both clauses of the predicate.

One problem with the linear inequality representation, however, is that arbitrarily large sets of inequalities can arise as the analysis proceeds. This can impede termination. Widening is therefore employed to constrict the growth of the sets and enforce convergence of the iterates to those inequalities that are common to all iterations. To be precise, $I_3 = I_2 \bigtriangledown I_3{}'$ where

$$\begin{aligned} I_3{}' &= \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid r = 0 \wedge s = t\} \ \overline{\cup} \\ &\quad\ \{\langle 1 + r, s, 1 + t\rangle \in \mathbb{R}^3 \mid 0 \leq r \wedge r \leq 1 \wedge t = r + s\} \\ &= \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid r = 0 \wedge s = t\} \ \overline{\cup} \\ &\quad\ \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid 1 \leq r \wedge r \leq 2 \wedge t = r + s\} \\ &= \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid 0 \leq r \wedge r \leq 2 \wedge t = r + s\} \end{aligned}$$

The widening $I_2 \bigtriangledown I_3{}'$ basically derives those inequalities that are common to both $I_2$ and $I_3{}'$. More precisely, it selects those inequalities of $I_2$ that hold for $I_3{}'$.

Each iteration will generate a space that is described by the set of inequalities. Until the widening is initiated, successive iterations will typically yield a space that both includes and extends the previous space. Intuitively, the invariant condition will be an expression of those spatial boundaries that are common between iterations. Those inequalities that are excluded, by widening, will be those that relate to variables whose size increases with each iteration and, in this case, represents the unconstrained growth of an argument that is a list. Once widening commences, termination follows since the set of inequalities at each iteration cannot grow any further. For this iteration, $I_3 = I_2 \bigtriangledown I'_3 = \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid 0 \leq r \wedge t = r + s\}$. Similarly, it can be shown that $I_4 = \{\langle r, s, t\rangle \in \mathbb{R}^3 \mid 0 \leq r \wedge t = r + s\}$, and hence the iteration sequence converges.

## 3 Preliminaries

### 3.1 Concrete Semantics

To express the widening and explain the implementation it is helpful to clarify the semantics. The semantics of the abstract program (and the concrete program) can be expressed in an $s$-style semantics for constraint logic programs [5]. The semantics is parameterised over an algebraic structure, $C$, of constraints. We write $c \models c'$ iff $c$ entails $c'$ and $c = c'$ iff $c \models c'$ and $c' \models c$. The interpretation base $B_C$ for the language defined by a program $P$ is the set of unit clauses of the form $p(\mathbf{x}) \leftarrow c$ quotiented by equivalence. Equivalence, $\sim$, is defined by: $p(\mathbf{x}) \leftarrow c \sim p(\mathbf{x}') \leftarrow c'$ iff $c \upharpoonright var(\mathbf{x}) = (c' \wedge (\mathbf{x} = \mathbf{x}')) \upharpoonright var(\mathbf{x}')$ where $\upharpoonright$ denotes projection. When $C$ corresponds to the Herbrand universe $Herb$, for example, $\sim$ is variance. The fixpoint semantics $\mathcal{F}_C$ is defined, as usual, in terms of an immediate consequences operator like so: $\mathcal{F}_C[\![P]\!] = lfp(T_P)$.

**Definition 1 fixpoint $s$-semantics for CLP.** The immediate consequences operator $T_P : B_C \rightarrow B_C$ is defined by:

$$T_P(I) = \left\{ [p(\mathbf{x}) \leftarrow c']_\sim \left| \begin{array}{ll} w \in P \quad \wedge w = p(\mathbf{t}) \leftarrow c, p_1(\mathbf{t}_1), \ldots, p_n(\mathbf{t}_n) & \wedge \\ [w_i]_\sim \in I \wedge w_i = p_i(\mathbf{x}_i) \leftarrow c_i & \wedge \\ \forall i. var(w) \cap var(w_i) = \emptyset & \wedge \\ \forall i \neq j. var(w_i) \cap var(w_j) = \emptyset & \wedge \\ c' = \wedge_{i=1}^{n} (\mathbf{x}_i = \mathbf{t}_i \wedge c_i) \wedge (\mathbf{x} = \mathbf{t}) \wedge c & \end{array} \right. \right\}$$

□

### 3.2 Abstract Semantics

*Ordering* A preorder is a preordered set $L$ ($\sqsubseteq$) where the relation $\sqsubseteq$ is reflexive and transitive. A poset is a preorder $L$ ($\sqsubseteq$) where $\sqsubseteq$ is also antisymmetric. A cpo is a complete poset, that is, ant $\mathbb{N}$-termed increasing chain $x_i \in L$ has a least upper bound $\sqcup_{i=1} x_i \in L$.

*Polyhedral Domains* Let $Lin$ denote the set of finite sets of implicitly conjoined non-strict inequalities. $Lin$ $(\models)$ is a preorder but lifts to a cpo $Lin/{=}$ $(\models)$ with quotienting. Let $Poly^n$ denote the set of (closed) polyhedral convex sets in $\mathbb{R}^n$. $Poly^n(\subseteq)$ is also a cpo. Given a finite, ordered set of variables $X = \{x_1, \ldots, x_n\}$, there is a natural mapping from $Lin/{=}$ to $Poly^n$ that is $poly_X([c]_=) = \{\mathbf{x} \in \mathbb{R}^n \mid (\bigwedge_{i=1}^n x_i = x_i') \models c\}$.

The preordering on inequalities lifts to interpretations to define a preorder $\wp(B_{Lin})$ $(\sqsubseteq)$ where $I \sqsubseteq I'$ iff $\forall [p(\mathbf{x}) \leftarrow c]_\sim \in I \,.\, \exists [p(\mathbf{x}) \leftarrow c']_\sim \in I' \,.\, c \models c'$. The preorder defines an equivalence relation: $I \approx I'$ iff $I \sqsubseteq I'$ and $I' \sqsubseteq I$ which, in turn, defines the poset $\wp(B_{Lin})/{\approx}$ $(\sqsubseteq)$ where $[I]_\approx \sqsubseteq [I']_\approx$ iff $I \sqsubseteq I'$. In fact $\wp(B_{Lin})/{\approx}$ $(\sqsubseteq)$ is a cpo. $T_P$ lifts to $\wp(B_{Lin})/{\approx}$ $(\sqsubseteq)$ by $T_P([I]_\approx) = [T_P(I)]_\approx$ and is continuous.

*Abstract Interpretation* Rather than adopt the Galois connection approach to abstract interpretation [8] we require widening to obtain stability of our fixpoint calculation, because the domain does not satisfy the ascending chain property.

**Definition 2 widening.** A widening $\triangledown$ on the preorder $L$ $(\sqsubseteq)$ is an operator $\triangledown : L \times L \rightarrow L$ such that: $\forall x, y \in L \,.\, x \sqsubseteq x \triangledown y$ and $\forall x, y \in L \,.\, y \sqsubseteq x \triangledown y$ and for all increasing chains $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$, the increasing chain defined by $y_0 = x_0, \ldots, y_{i+1} = y_i \triangledown x_{i+1}, \ldots$ is not strictly increasing, that is, $y_{l+1} \sqsubseteq y_l$ for some $l$. □

To improve precision we adapt the widening strategy of [8] and only apply the operator after a bounded number of iterations.

**Proposition 3 adapted from [8].** If $L$ $(\sqsubseteq, \sqcup)$ is a cpo, $F : L \rightarrow L$ is continuous, $\perp \in L$ is such that $\perp \sqsubseteq F(\perp)$, $\triangledown \in L \times L \rightarrow L$ is a widening, then the upward iteration sequence with widening $x_i$ where $i, k \in \mathbb{N}$ is defined thus:

$$\begin{aligned} x_0 &= \perp \\ x_{i+1} &= x_i && \text{if } F(x_i) \sqsubseteq x_i \\ x_{i+1} &= F(x_i) && \text{else if } i \leq k \\ x_{i+1} &= x_i \triangledown F(x_i) && \text{else if } i > k \end{aligned}$$

will converge and its limit $\mathcal{A}$ is such that $lfp(F) \sqsubseteq \mathcal{A}$ and $F(\mathcal{A}) \sqsubseteq \mathcal{A}$. □

### 3.3 Argument Size Analysis

A concretisation mapping is used to clarify the relationship between a concrete and abstract program in terms of a norm $|t|$ that measures the size of a term $t$.

**Definition 4 $\gamma$.** Concretisation $\gamma(I) : \wp(B_{Lin}) \rightarrow \wp(B_{Herb})$ is defined by:

$$\gamma(I) = \left\{ [p(\mathbf{x}) \leftarrow \textstyle\bigwedge_{i=1}^n (x_i = t_i)]_\sim \;\middle|\; \begin{array}{l} [p(\mathbf{x}) \leftarrow c]_\sim \in I \quad \wedge \\ (\bigwedge_{i=1}^n x_i = |t_i|) \models c \end{array} \right\}$$

□

A program $P$ over $Herb$ is safely abstracted by abstract program $P^{\mathcal{A}}$ over $Lin$ iff $\mathcal{F}_{Herb}[\![P]\!] \subseteq \gamma(\mathcal{F}_{Lin}[\![P^{\mathcal{A}}]\!])$.

## 4 Convex Hull Calculation

Previous approaches [9, 15, 16, 22, 32] to computing the convex hull of polyhedra rely on the frame representation. Specifically, the polyhedra are represented as a system of generators, that is, two finite sets, $V$ and $R$, of vertices and rays:

$$P = \left\{ \sum_{v_i \in V} \lambda_i . v_i + \sum_{r_j \in R} \mu_j . r_j \;\middle|\; \lambda_i \geq 0 \wedge \mu_j \geq 0 \wedge \sum_i \lambda_i = 1 \right\}$$

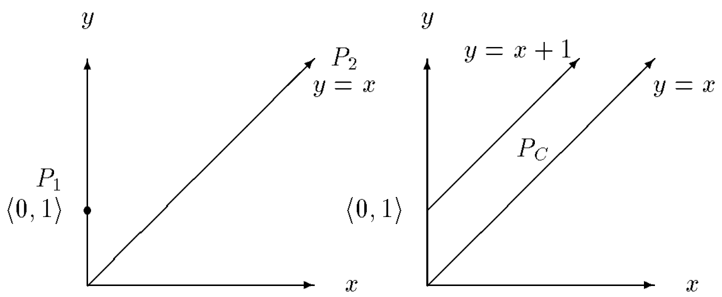
The convex hull $P$ of two polyhedra $P_1$ and $P_2$, respectively represented by $\langle V_1, R_1 \rangle$ and $\langle V_2, R_2 \rangle$, is then given by $\langle V, R \rangle$ where $V = V_1 \cup V_2$ and $R = R_1 \cup R_2$

*Example 1.* Consider the point $P_1$ and the line $P_2$. The convex hull of $P_1$ and $P_2$ is the space $P_C$. Both the constraint and frame representations of $P_1$, $P_2$ and $P_C$ are given below followed by two graphs that depict the polyhedra.

$$P_1 = \left\{ \langle x, y \rangle \in \mathbb{R}^2 \;\middle|\; \begin{array}{l} x \leq 0 \wedge \\ 0 \leq x \wedge \\ y \leq 1 \wedge \\ 1 \leq y \end{array} \right\} \qquad V_1 = \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \qquad R_1 = \emptyset$$

$$P_2 = \left\{ \langle x, y \rangle \in \mathbb{R}^2 \;\middle|\; \begin{array}{c} x \leq y \;\; \wedge \\ y \leq x \;\; \wedge \\ -x \leq 0 \end{array} \right\} \quad V_2 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} \qquad R_2 = \left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$

$$P_C = \left\{ \langle x, y \rangle \in \mathbb{R}^2 \;\middle|\; \begin{array}{c} x - y \leq 0 \wedge \\ y - x \leq 1 \wedge \\ -x \leq 0 \end{array} \right\} V_C = \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} R_C = \left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$




□

Usually the constraints and frame are represented together and the Chernikova algorithm is used to convert between them. For example to compute an over approximation of the convex hull, $V_C$ and $R_C$ are computed and then the

Chernikova algorithm is used to generate $P_C$. Both representations are used simultaneously as "experience shows that this redundant representation is much less expensive than the frequent use of conversions" [9]. It is interesting to note that it is the closure of the convex hull that is returned by both methods, that is the smallest polyhedral convex set that includes the convex hull.

By using a different approach to computing the convex hull, it is possible to use a single representation, namely a set of linear inequalities. CLP($\mathcal{R}$) provides the projection and solver machinery for manipulating sets of inequalities and thus allows us to implement the convex hull in an efficient but relatively simple way. The naive approach to the calculating the convex hull in CLP($\mathcal{R}$) can lead to floundering. Floundering occurs because non-linear constraints may be indefinitely postponed. Suppose that two arbitrary polyhedra, $P_1$ and $P_2$, are represented in standard form, that is,

$$P_1 = \{\mathbf{X} \in \mathbb{R}^n \mid A_1\mathbf{X} \leq \mathbf{B}_1\}, \quad P_2 = \{\mathbf{X} \in \mathbb{R}^n \mid A_2\mathbf{X} \leq \mathbf{B}_2\}$$

The convex hull of $P_1 \cup P_2$, $P$, is then defined by:

$$P_C = \left\{ \mathbf{X} \in \mathbb{R}^n \;\middle|\; \begin{array}{ll} \mathbf{X} = \sigma_1\mathbf{X}_1 + \sigma_2\mathbf{X}_2 & \wedge\ \sigma_1 + \sigma_2 = 1 \ \wedge \\ A_1\mathbf{X}_1 \leq \mathbf{B}_1 & \wedge\ A_2\mathbf{X}_2 \leq \mathbf{B}_2 \ \wedge \\ -\sigma_1 \leq 0 & \wedge\ \ -\sigma_2 \leq 0 \end{array} \right\}$$

The equation $\sigma_1\mathbf{X}_1 + \sigma_2\mathbf{X}_2 = 1$, however, is non-linear and in a constraint language that delays non-linear constraints the worst case can result in an infinite loop [17]. Following [10], however, equations can be reformulated by putting $\mathbf{Y}_1 = \sigma_1\mathbf{X}_1$ and $\mathbf{Y}_2 = \sigma_2\mathbf{X}_2$ so that

$$\mathbf{X} = \mathbf{Y}_1 + \mathbf{Y}_2, \quad A_1\mathbf{Y}_1 \leq \sigma_1\mathbf{B}_1, \quad A_2\mathbf{Y}_2 \leq \sigma_2\mathbf{B}_2$$

so that $P_C$ is also defined by:

$$P_C = \left\{ \mathbf{X} \in \mathbb{R}^n \;\middle|\; \begin{array}{c} \mathbf{X} = \mathbf{Y}_1 + \mathbf{Y}_2 \wedge \ \ \sigma_1 + \sigma_2 = 1 \ \ \wedge \\ A_1\mathbf{Y}_1 \leq \sigma_1\mathbf{B}_1 \wedge A_2\mathbf{Y}_2 \leq \sigma_2\mathbf{B}_2 \wedge \\ -\sigma_1 \leq 0 \quad \wedge \quad -\sigma_2 \leq 0 \end{array} \right\}$$

*Example 2.* To illustrate the method, we refer to our earlier example. Substituting for the matrices $A_1$and $A_2$, and the vectors $B_1$ and $B_2$, the above system of equations is as follows:

$$P_C = \left\{ \mathbf{X} \in \mathbb{R}^n \;\middle|\; \begin{array}{ccc} \mathbf{X} = \mathbf{Y}_1 + \mathbf{Y}_2 & \wedge & \sigma_1 + \sigma_2 = 1 \quad \wedge \\ \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \mathbf{Y}_1 \leq \sigma_1 \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix} & \wedge & \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{Y}_2 \leq \sigma_2 \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \ \wedge \\ -\sigma_1 \leq 0 & \wedge & -\sigma_2 \leq 0 \end{array} \right\}$$

Note that $\mathbf{X} = \langle x^1, x^2 \rangle$ and $\mathbf{Y}_i = \langle y_i^1, y_i^2 \rangle$,

$$P_C = \left\{ \mathbf{X} \in \mathbb{R}^2 \;\middle|\; \begin{array}{lcr} x^1 = y_1^1 + y_1^2 & \wedge & \sigma_1 + \sigma_2 = 1 \wedge \\ x^2 = y_2^1 + y_2^2 & \wedge & -\sigma_1 \leq 0 \wedge \\ y_1^1 \leq 0 & \wedge & -\sigma_2 \leq 0 \wedge \\ -y_1^1 \leq 0 & \wedge & y_2^1 - y_2^2 \leq 0 \wedge \\ y_1^2 \leq \sigma_1 & \wedge & -y_2^1 + y_2^2 \leq 0 \wedge \\ -y_1^2 \leq -\sigma_1 & \wedge & -y_2^1 \leq 0 \end{array} \right\}$$

and hence $P_C$ can be derived through projection. □

In terms of implementation, the chief technicality is in constructing the equations $A_i\mathbf{Y}_i \leq \sigma_i\mathbf{B}_i$ from $A_i\mathbf{X}_i \leq \mathbf{B}_i$. In fact each $A_i\mathbf{Y}_i \leq \sigma_i\mathbf{B}_i$ can be generated by a single recursive pass over the ground representation of $A_i\mathbf{X}_i \leq \mathbf{B}_i$ which basically collects and then scales the numeric constants. Once the equations are setup, a projection onto $\mathbf{X}$ then gives $P_C$ encoded in a ground representation.

## 5 Widening Operation

Widening is required to enforce the convergence of the iterates. Essentially it trades precision for finiteness by weakening inequality constraints to obtain stability of the iterates. Since $T_P$ is continuous on the cpo $\wp(B_{Lin})/\approx (\sqsubseteq)$, $\perp \sqsubseteq T_P(\perp)$ where $\perp = [\emptyset]_\approx$, by Proposition 3 it only remains to define a suitable $\triangledown$ operator for $\wp(B_{Lin})/\approx$ and to select an appropriate $k$. The widening $[I]_\approx \triangledown [I']_\approx$ is basically an adaption of the widening of [9] lifted to interpretations. Since $I$ and $I'$ both contain at most one set of inequalities for each predicate symbol, the widening lifts in a straightforward way.

**Definition 5.**

$$[I]_\approx \triangledown [I']_\approx = [\{[p(\mathbf{x}) \leftarrow c \triangledown c']_\sim \,|[p(\mathbf{x}) \leftarrow c]_\sim \in I \wedge [p(\mathbf{x}) \leftarrow c']_\sim \in I' \}]_\approx$$

$$\text{where } c \triangledown c' = \{ i \in c \mid c' \models i \}$$

□

The widening $c \triangledown c'$ relaxes the constraint $c$ by selecting those inequalities $i$ of $c$ which are entailed by $c'$. Since $c$, $c'$ are encoded in the ground representation, the test for entailment amounts to scanning the list of constraints representing $c$ and then testing each $i$ in the list against $c'$ with entailment. Termination follows since the widening stops the set (list) representing $c'$ including new inequalities. (Interestingly, we have found that the naive widening $c \triangledown c' = c \cap c'$ can work well if the constraints always appear in the same syntactic form.) The main subtlety in widening is choosing a useful $k$, that is, deciding *when* to widen. Sections 5.1, 5.2 and 5.3 explain how $k$ affects the precision for different classes of predicate.

### 5.1 Widening with Uniform Increments

Consider the $\texttt{Ap}^{\mathcal{A}}\texttt{/3}$ program of Section 2 listed below.

```
Ap^A(0, s, s).
Ap^A(1 + r, s, 1 + t) <- Ap^A(r, s, t).
```

Each iteration of the analysis generates an atom $\texttt{Ap}^{\mathcal{A}}(x, y, z) \leftarrow c$ where the variables $x$ and $z$ are both incremented by 1 relative to the previous iterate. To be more precise, the $i^{th}$ iteration of the analysis, $[I_i]_{\approx}$, takes the form $I_i = \{[\texttt{Ap}^{\mathcal{A}}(x, y, z) \leftarrow c_i]_{\sim}\}$ where each $c_i$ defines a polyhedral convex set in $\mathbb{R}^3$ by $poly_{\{x,y,z\}}(c_i) = p_i$. For example

$$p_1 = \{\langle x, y, z\rangle \mid x = 0, y = z\}, \quad p_2 = \{\langle x, y, z\rangle \mid 0 \leq x, x \leq 1, z = x + y\}$$

More generally $\texttt{Ap}^{\mathcal{A}}\texttt{/3}$ defines $p_{i+1} = p_i \overline{\cup} \{\langle 1+x, y, 1+z\rangle \mid \langle x, y, z\rangle \in p_i\}$ so that the space $p_{i+1}$ extends and includes that of $p_i$. Each $p_{i+1}$ can be obtained from $p_i$ in a predictable way since $p_{i+1}$ differs from $p_i$ by uniform increments in the first and third dimensions. Although inter-argument relationships $0 \leq x, z = x + y$ are implicit in $p_1$, they are not explicit until $p_2$ and the ensuing $p_i$. Widening can therefore be performed to obtain the third iterate, that is $k = 2$, without loss of significant information. The invariant condition is then confirmed in the third iteration to obtain $p_3 = \{\langle x, y, z\rangle \mid 0 \leq x, z = x+y\}$. Widening prematurely looses information. We conjecture that for a directly recursive predicate with uniform increment, all of the common invariants can be found within three iterations.

### 5.2 Widening within a Hierarchy

Consider the $\texttt{Qs}^{\mathcal{A}}$ program of section A. The program consists of a hierarchy of several predicates, where the top level predicate $\texttt{Qs}^{\mathcal{A}}/2$ has calls in its body to other predicates, the auxiliaries $\texttt{Pt}^{\mathcal{A}}/4$ and $\texttt{Ap}^{\mathcal{A}}/3$. Each $I_i$ therefore will consist of possibly many $[p(\mathbf{x}) \leftarrow c]_{\sim}$, at most one for each predicate symbol $p$. $I_{i+1}$ can only include $[\texttt{Qs}^{\mathcal{A}}(\mathbf{x}) \leftarrow c]_{\sim}$, however, provided $I_i$ includes $[\texttt{Pt}^{\mathcal{A}}(\mathbf{x}) \leftarrow c]_{\sim}$ and $[\texttt{Ap}^{\mathcal{A}}(\mathbf{x}) \leftarrow c]_{\sim}$. $\texttt{Pt}^{\mathcal{A}}$ and $\texttt{Ap}^{\mathcal{A}}$ are directly recursive with uniform increment and therefore can be widened with $k = 2$. In general, however, precision can be lost if a predicate is widened before its auxiliaries are widened and thus stable. By inspecting the clauses of a program, the call graph and its SCCs can be computed. The SCCs of the $\texttt{Qs}^{\mathcal{A}}$ program, for example, are the clause sets $\{\{\texttt{Ap}^{\mathcal{A}}/3/1\}$, $\{\texttt{Ap}^{\mathcal{A}}/3/2\}$, $\{\texttt{Pt}^{\mathcal{A}}/4/1\}$, $\{\texttt{Pt}^{\mathcal{A}}/4/2, \texttt{Pt}^{\mathcal{A}}/4/3\}$, $\{\texttt{Qs}^{\mathcal{A}}/2/1\}$, $\{\texttt{Qs}^{\mathcal{A}}/2/2\}\}$ where the $p/n/m$ notation abbreviates the $m^{th}$ clause defining the predicate $p/n$. SCCs can be used to compute the fixpoint in a bottom-up fashion by considering the SCCs in (reverse) topological order. See Figure 1. Analysis begins with the base cases of the deepest predicates, and progressing upwards to derive fixpoints for $\texttt{Pt}^{\mathcal{A}}$ and $\texttt{Ap}^{\mathcal{A}}$, before moving on to $\texttt{Qs}^{\mathcal{A}}$. A complete analysis for $\texttt{Qs}^{\mathcal{A}}$ is given in the table.

| *Interpretation* | *Step* |
|---|---|
| $I_1 = \{[\mathtt{Ap}^{\mathcal{A}}(x,y,z) \leftarrow x = 0, y = z]_{\sim}\}$ | base |
| $I_2 = \{[\mathtt{Ap}^{\mathcal{A}}(x,y,z) \leftarrow 0 \leq x \leq 1, z = y + x]_{\sim}\}$ | recurse |
| $I_3 = \{[\mathtt{Ap}^{\mathcal{A}}(x,y,z) \leftarrow z = y + x, (-x) \leq 0]_{\sim}\}$ | widen |
| $I_4 = \{[\mathtt{Ap}^{\mathcal{A}}(x,y,z) \leftarrow z = y + x, (-x) \leq 0]_{\sim}\}$ | stabilise |
| $I_5 = I_4 \cup \{[\mathtt{Pt}^{\mathcal{A}}(w,x,y,z) \leftarrow x = 0, y = 0, z = 0]_{\sim}\}$ | base |
| $I_6 = I_4 \cup \{[\mathtt{Pt}^{\mathcal{A}}(w,x,y,z) \leftarrow x = z + y, 0 \leq y \leq 1, y - x \leq 0]_{\sim}\}$ | recurse |
| $I_7 = I_4 \cup \{[\mathtt{Pt}^{\mathcal{A}}(w,x,y,z) \leftarrow x = z + y, y \leq x, (-y) \leq 0]_{\sim}\}$ | widen |
| $I_8 = I_4 \cup \{[\mathtt{Pt}^{\mathcal{A}}(w,x,y,z) \leftarrow x = z + y, y \leq x, (-y) \leq 0]_{\sim}\}$ | stabilise |
| $I_9 = I_8 \cup \{[\mathtt{Qs}^{\mathcal{A}}(x,y) \leftarrow x = 0, y = 0]_{\sim}\}$ | base |
| $I_{10} = I_8 \cup \{[\mathtt{Qs}^{\mathcal{A}}(x,y) \leftarrow y = x, 0 \leq x \leq 1]_{\sim}\}$ | recurse |
| $I_{11} = I_8 \cup \{[\mathtt{Qs}^{\mathcal{A}}(x,y) \leftarrow y = x, (-x) \leq 0]_{\sim}\}$ | widen |
| $I_{12} = I_8 \cup \{[\mathtt{Qs}^{\mathcal{A}}(x,y) \leftarrow y = x, (-x) \leq 0]_{\sim}\}$ | stabilise |

Note that whilst other topological orderings are possible, this particular ordering minimises the size of the interpretations and thus simplifies the presentation. Widening each SCC separately also improves the efficiency. Since a predicate will not be referenced until its auxiliaries have been analysed, fewer computations will lead to failure. (Interestingly, the difference equation approach of [11] is unable useful inter-argument relations for divide and conquer methods like $\mathtt{Qs}^{\mathcal{A}}$ in the way our approach does.)

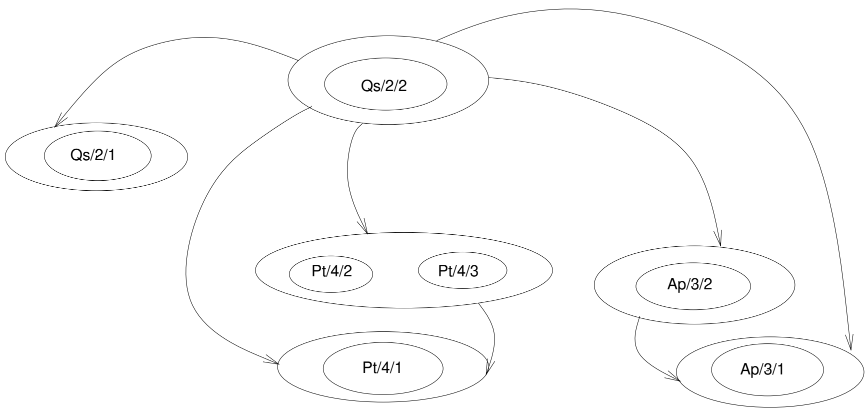



**Fig. 1.** Dependencies between SCCs of the $\mathtt{Qs}^{\mathcal{A}}$ program

### 5.3 Widening with Non-uniform Increments

However under certain conditions all pertinent information may not be found in three iterations even when the SCCs are considered one by one. Through experimentation with our analyser, we have identified two classes of predicate that require a more intelligent widening strategy. Consider the $\mathtt{Sp}^{\mathcal{A}}/3$ predicate introduced in Section 1.

```
Sp^A(0, 0, 0).
Sp^A(1+xs, 1+os, es) <- Sp^A(xs, es, os).
```

Here elements of the first list are placed alternately in the second and third lists. Thus $\mathtt{Sp}^{\mathcal{A}}/3$ defines

$$p_{i+1} = p_i \,\overline{\cup}\, \{\langle 1+x, 1+z, y\rangle \mid \langle x, y, z\rangle \in p_i\},$$
$$p_{i+2} = p_i \,\overline{\cup}\, \{\langle 2+x, 1+y, 1+z\rangle \mid \langle x, y, z\rangle \in p_i\}$$

so that $\mathtt{Sp}^{\mathcal{A}}/3$ has a bi-modal incrementation behaviour where a uniform increment of 2, 1 and 1 in the first, second and third dimensions occurs in every second iteration. Consequently the invariant condition cannot be confirmed until the fifth iteration. Put another way, $k = 4$ is required. An analysis for $\mathtt{Sp}^{\mathcal{A}}$ is shown below where $I_i = \{[\mathtt{Sp}^{\mathcal{A}}(x, y, z) \leftarrow c_i]_\sim\}$. The constraints in sans serif type are those that comprise the invariant condition.

| | | | | | | |
|---|---|---|---|---|---|---|
| $c_1$ | $x = 0$ | $y = 0$ | $z = 0$ | | | |
| $c_2$ | $x \leq 1$ | $-x \leq 0$ | $z = 0$ | $y - x = 0$ | | |
| $c_3$ | $\mathsf{x - y - z = 0}$ | $\mathsf{-y \leq 0}$ | $\mathsf{x - 2y \leq 0}$ | $y - x \leq 0$ | $y \leq 1$ | |
| $c_4$ | $\mathsf{x - y - z = 0}$ | $\mathsf{-y \leq 0}$ | $\mathsf{x - 2y \leq 0}$ | $\mathsf{2y - x \leq 1}$ | $\mathsf{y - x \leq 0}$ | $x - y \leq 1$ |
| $c_5$ | $\mathsf{x - y - z = 0}$ | $\mathsf{-y \leq 0}$ | $\mathsf{x - 2y \leq 0}$ | $\mathsf{2y - x \leq 1}$ | $\mathsf{y - x \leq 0}$ | |

The widening as described in [8] would converge after three iterations having only deduced $x - y - z = 0$.

Another related problem is illustrated by the `St/3` predicate, expressing a step function, listed below.

```
St(0, 0).
St(x + 1, 0) <-
   x =< 2, St(x, y).
St(x + 1, 1) <-
   2 < x, St(x, y).
```

```
St^A(0, 0).
St^A(x + 1, 0) <-
   x =< 2, St^A(x, y).
St^A(x + 1, 1) <-
   2 < x, St^A(x, y).
```

The abstract version $\mathtt{St}^{\mathcal{A}}$ is coincident with `St/3` when the norm is size of an integer. An analysis for $\mathtt{St}^{\mathcal{A}}$ is shown below where $I_i = \{[\mathtt{St}^{\mathcal{A}}(x, y) \leftarrow c_i]_\sim\}$.

| | | | | |
|---|---|---|---|---|
| $c_1$ | $x = 0$ | $y = 0$ | | |
| $c_2$ | $\mathsf{0 \leq x}$ | $x \leq 1$ | $y = 0$ | |
| $c_3$ | $\mathsf{0 \leq x}$ | $x \leq 2$ | $y = 0$ | |
| $c_4$ | $\mathsf{0 \leq x}$ | $x \leq 3$ | $\mathsf{0 \leq y}$ | $\mathsf{y \leq 1}$ |
| $c_5$ | $\mathsf{0 \leq x}$ | $\mathsf{0 \leq y}$ | $\mathsf{y \leq 1}$ | |

Note that third clause of $\mathtt{St}^{\mathcal{A}}$ is only selected in the fourth iteration so premature widening leads to loss of precision. More generally, precision may be lost when the selection of a clause is postponed until a threshold is reached. It is therefore apparent that the moment at which to apply the widening is when the iterations have captured all the invariant relationships. As shown above, however, detecting this moment may present some difficulties and therefore the current implementation permits the user to vary $k$. The problem of choosing when to widen relates to the usual trade-off between cost and accuracy.

## 6 Implementation

### 6.1 Ground (and Non-ground) Representation

A ground representation is used in combination with a non-ground representation. This simplifies the meta-interpreter as atoms can be looked up in the interpretation without inducing aliasing. The interpretations are ground whereas the programs are non-ground.

### 6.2 Bottom-up Interpreter

The bottom-up analyser is basically a simplified version of the meta-interpreter listed in [6]. The main difference between the two analysers is that our interpretations are ground and therefore variance (goal renaming) is not an issue. For a given clause, the interpreter essentially unifies each (non-ground) body atom in the definition of the clause with a matching (ground) atom in the interpretation, and then projects onto the head. Builtins are solved directly. The task of unifying a body atom with an atom in the interpretation basically reduces to decoding the ground representation of constraints. For example, if the body atom is `p(U, V, W)` and the interpretation contains `atom(p, 3, [less(plus(var(1), var(2)), 0)])` then the call to decode from the ground representation

```
decode([less(plus(minus(var(1)), var(2)), 0)], 3, [U, V, W])
```

imposes the constraint `V < U` on the constraint store. The constraints on the head variables are then projected using meta-programming builtins to obtain an output in a ground form. Polyhedral abstractions from each of the clauses of a predicate are then combined incrementally as a convex hull. The convex hull operation is binary so for a predicate of say three clauses yielding the polyhedra $p_1, p_2, p_3$, the hull can be calculated by $(p_1 \overline{\cup} p_2) \overline{\cup} p_3$ with no loss of precision since the operation is both associative and commutative. Projection is used to eliminate the slack variables introduced in the convex hull calculation.

### 6.3 Entailment in CLP($\mathcal{R}$) and SICStus 3

The widening operation identifies those constraints that are invariant between iterations. At the implementation level this involves a test for entailment which is based upon the premise that for a linear constraint $c$ to be entailed by a set of constraints $C$ it is sufficient to show that $C \wedge \neg c$ has no solution [7].

### 6.4 Projection in CLP($\mathcal{R}$)

The constraints on the head variables are projected using the CLP($\mathcal{R}$) meta-programming built-in `dump/3` [17] and the projection is output in a ground form. For example, with the constraint store $\mathtt{X} = 4 + \mathtt{Y}, \mathtt{Z} < \mathtt{X}$, the call

```
dump([X, Y], [var(1), var(2)], Cons)
```

instantiates `Cons` to `[var(2) = var(1) - 4]` which corresponds to a ground representation of the projection onto `X` and `Y`. To improve portability `dump/3` is the only meta-programming facility that was utilised in CLP($\mathcal{R}$).

### 6.5 Projection in SICStus 3

The `call_residue/2` built-in in SICStus has a similar role to `dump/3` in CLP($\mathcal{R}$). Apart from small syntactic differences, the projection machinery is the main place where the CLP($\mathcal{R}$) and SICStus 3 analysers differ. In SICStus, projection is rather more complex as inequalities and equalities have to be dealt with separately. `call_residue/2` accesses the residual constraints, that is, those constraints that are not satisfiable at the time of calling. For example, the call

```
call_residue({X = Y + Z, X = P}, Cons)
```

instantiates `Cons` to `[[Z]-Z = P-Y]`. Note, however, that `Cons` excludes the equality relationship between `X` and `P`. Since `call_residue/2` is called with a goal, the program is dynamically modified with the call

```
assert(dummy(Target_Vars))
```

where `Target_Vars` are the variables that are the projection target. A `dummy/1` goal is then passed to `call_residue/2` to retrieve the constraints. A final post-processing phase assembles the residual constraints with the equality constraints.

### 6.6 Arithmetic in CLP($\mathcal{R}$) and SICStus 3

The coefficients of variables within the inequalities are represented as real numbers so, for example, the inequality $2x \leq 3y$ may sometimes be stored as $x - 1.5y \leq 0$ and as $x - 1.499999999999y \leq 0$ in another iteration. Although the widening ensures that termination is not compromised, roundoff errors can lead to some interesting relationships being lost. Roundoff is not a problem in the CLP($\mathcal{R}$) implementation since CLP($\mathcal{R}$) uses an small $\epsilon$ as slack in its numerical comparisons. Roundoff, however, lost relationships in the `Trd/3` problem (see Appendix A) in the initial SICStus 3 clp(R) implementation and and this hastened the port to clp(Q).

## 7 Related Work

*Most Specific Generalisation (msg) Approximations* Argument size analysis can be specified by defining an appropriate pre-interpretation and by using *msg* approximations [13]. *msg* approximations are formed by taking most specific generalisations of sets of atoms. With the aid of query-answer transforms, the analysis of [13] can infer that the length of the first argument of naive reverse `rev/2` is the same as the length of the second argument. Inequalities are not supported.

*Affine Approximations* In the affine approach [14, 20, 31], invariants are represented by affine subspaces, basically points, lines or hyper-planes in $\mathbb{R}^n$, which can be represented and manipulated using matrices. The affine approach is attractive because, although it cannot infer inequalities between variables, the approximation is Noetherian and therefore the termination of the analysis is not an issue.

*Simple Section Approximations* Although developed for imperative languages, we suspect that simple sections [4] could be used to express simple inter-argument relationships. Simple sections characterise spaces that are bounded by hyper-planes of the form $x = c$, $x + y = c$ or $x - y = c$ where $x$ and $y$ are points on two different co-ordinate axes and $c$ is a constant (or a special $-\infty$ or $+\infty$ symbol [8]). Simple sections can potentially reduce the complexity of convex hull calculations, can be implemented easily without CLP($\mathcal{R}$) support, but are likely to sacrifice a lot of precision. Moreover, the representation is likely to become intractable for predicates of larger arity.

*Interval Approximations* CHINA [1, 2, 3] is an analyser for CLP($\mathcal{R}$) and CLP($\mathcal{FD}$) that approximates conjunctions of constraints with bounding boxes. Bounding boxes are rectangular regions with sides parallel to the axes that are obtained by projecting variables onto their axes in order to represent each variable with the interval that results from the projection. Interval widening is required for termination and constraints are solved by propagating constraints around a constraint network [1, 2]. The bounding box approximation has been chosen for tractability, but our work suggests that, with some thought, it is not difficult to implement a more precise analysis with polyhedral approximations. To be fair, however, CHINA [2] introduces some nice ideas like, for example, the way constraints are compiled into finite concurrent constraint agents over a product of domains to trace the behaviour of builtins.

An interval analysis for CLP($\mathcal{R}$), not dissimilar to that used in CHINA, is described in [19]. The analysis is coupled to GAIA, is implemented in C, and uses narrowing to recover some of the precision lost in widening interval approximations. Finally, although not implemented, the paper explains how intervals can be coupled with groundness descriptions.

*Polyhedral Approximations* In [24] a Prolog III program is presented for checking the invariants of the CLP($\mathcal{Q}$) program to partially mechanise the derivation of inter-argument relationships. The proof method is neither a decision procedure (it is not complete) nor is it automatic since "its main drawback comes from the fact that the user has to provide the linear inter-argument relation to be proven".

In [30], an analysis for inferring linear inequalities is proposed with a suite of matrix transformations for mechanising the derivation of the inequalities. One flaw of [30], however, is that the iterative process required to compute inequalities may not converge in finitely many steps. Therefore Van Gelder proposes "an heuristic [for finding fixpoints] which often works" [30] and otherwise has recourse to human intervention, noting that useful fixpoints are not easy to find.

A widening for polyhedra is reported in [15, 16] that refines a widening first proposed in [9]. Originally used for overflow and array bounds checking, these widenings essentially remove inequalities from the first polyhedron that are not satisfied by the second. So that less information is lost the widening of [15, 16] reformulates the representation of the first polyhedron in order to maximise the number of common constraints. For example, to calculate $P_1 \bigtriangledown P_2$ where $P_1 = \{\langle x, y\rangle \in \mathbb{R}^2 \mid x = 0, y = 0\}$ and $P_2 = \{\langle x, y\rangle \in \mathbb{R}^2 \mid 0 \leq y \leq x \leq 1\}$, $P_1$ is re-expressed as $P_1 = \{\langle x, y\rangle \in \mathbb{R}^2 \mid 0 \leq y \leq x \leq 0\}$, yielding a result $P_1 \bigtriangledown P_2 = \{\langle x, y\rangle \in \mathbb{R}^2, \mid 0 \leq y \leq x\}$ rather than $\{\langle x, y\rangle \in \mathbb{R}^2 \mid 0 \leq x, 0 \leq y\}$. An algorithm for reformulating $P_1$ is detailed in [15].

Cousot and Cousot [8] address the termination problem of [30] by employing a widening to enforce convergence of the iterates. The widening builds on those reported in [9, 15, 16]. Given two polyhedral sets, $P_i \bigtriangledown P_{i+1}$, is computed from two sets of inequalities, $S_i$ and $S_{i+1}$, that define $P_i$ and $P_{i+1}$. First, the widening selects those inequalities of $S_i$ that hold for all the points in $P_{i+1}$. This subset of $S_i$ is denoted by ${S_i}'$. Second, the widening finds those inequalities which hold for $P_i$ and $P_{i+1}$ and yet are not explicit in $S_i$. For example, if $S_i = \{x = 1, y = 1\}$ and $S_{i+1} = \{x = y\}$, then $x = y$ holds for $P_i$ and $P_{i+1}$ but $x = y \notin S_i$. The widening thus selects those inequalities of $S_{i+1}$, like $x = y$, that can be swapped with an inequality of $S_i$, like $x = 1$, without altering the space $P_i$. This subset of $S_{i+1}$ is denoted by ${S_{i+1}}'$. The widening $P_i \bigtriangledown P_{i+1}$ is taken to be the polyhedron defined by ${S_i}' \cup {S_{i+1}}'$. The widening uses the frame representation [9].

The argument size analysis of [28] builds on the matrix transforms of [30] but also uses a so-called affine widening to ensure termination. The widening, however, looses much of the expressiveness of polyhedral approximations and it "cannot infer relationships in the form of inequality" [28]. To improve the precision (and avoid widening) an unfolding transform is proposed for the special class of linearly recursive programs. A frame representation is used to check for a fixpoint and the representation also seems to be used in the convex union calculation [28, Section 2].

## 8 Future Work

In terms of efficiency, we believe that it might be possible, under certain conditions, to simplify the convex hull operation. In terms of precision, the widening can sometimes be improved by a single narrowing step $P_1 \triangle P_2 = P_1 \wedge P_2$. Future work will quantify the usefulness of this refinement. Future work will also investigate how the shapes of successive polyhedra can be used to deduce the propitious moment to widen and characterise the class of predicates that can be widened quickly without losing precision. We will also investigate how SCCs relate to chaotic iteration with widening [8]. We also intend to examine how the analysis can be ported to other systems with the (meta-programming) facilities offered by other languages, for example, Gödel, CHIP and IF/Prolog.

## Acknowledgements

We should like to extend special thanks to Christian Holzbaur, for tirelessly and promptly coping with our queries regarding SICStus 3#3. Also, we gratefully acknowledge John Gallagher, Alain Kerbrat, Kung-Kiu Lau, Pedro Lopez Garcia, Jon Martin, Håkan Millroth, Julian Richardson, and Jan Smaus for their useful discussions. This work was generously supported by the Nuffield grant SCI/180/94/417/G and a Brian Spratt bursary.

## References

1. R. Bagnara. On the detection of implicit and redundant numeric constraints in CLP programs. In *GULP-PRODE'94*, pages 312–326, 1994.
2. R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. Technical Report TR-96-10, Dipartimento di Informaticá, Universita di Pisa corso Italia 40, 56125 Pisa, Italy, 1996.
3. R. Bagnara, R. Giacobazzi, and G. Levi. Static Analysis of clp Programs over Numeric Domains. In *WSA'92*, 1992.
4. V. Balasundaram and K. Kennedy. A Technique for Summarizing Data Access and Its Use in Parallelism Enhancing Transformations. In *PLDI'89*, pages 41–53. ACM Press, 1989.
5. A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The *s*-semantics approach: theory and applications. *Journal of Logic Programming*, 1991.
6. M. Codish and V. Lagoon. Persistant Type Analysis using a Non-Ground Domain. Technical report, Dept of Maths and Computer Science, Ben-Gurion University of the Negev, Israel, 1995.
7. A. Colmerauer. An Introduction to Prolog III. In *CACM*, volume 33, pages 70–90, July 1990.
8. 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 Normale Superiéure, 45 Rue d'Ulm, 75230 Paris Cédex 05, France, 1992.
9. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *POPL'78*, pages 84–97, 1978.

10. B. De Backer and H. Beringer. A CLP language handling disjunctions of linear constraints. In *ICLP'93*, pages 550–563. MIT Press, 1993.
11. S. Debray and N. Lin. Cost Analysis for Logic Programs. *ACM Transactions on Programming Languages and Systems*, July 1992.
12. S. Decorte, D. De Schreye, and M. Fabris. Automatic Inference of Norms: a Missing Link in Automatic Termination Analysis. In *ICLP'93*, pages 420–436, 1993.
13. J. Gallagher, D. Boulanger, and Y. Saglam. Practical Model-Based Static Analysis for Definite Logic Programs. Technical Report CSTR-95-011, Department of Computer Science, University of Bristol, June 1995.
14. R. Giacobazzi, S. Debray, and G. Levi. Generalised Semantics and Abstract Interpretation for Constraint Logic Programs. Technical report, Dipartimento di Informatica, Universitá di Pisa, 1992.
15. N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Universit'e scientifique et médicale de Grenoble, 1979. Thèse de 3 ème d'informatique.
16. N. Halbwachs, Y.E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *First International Static Analysis Symposium*. Springer Verlag, September 1994.
17. N. C. Heintze, J. Jafar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. *The CLP($\mathcal{R}$) Programmer's Manual Version 1.2*, 1992.
18. R. N. Horspool. Analyzing List Usage in Prolog Code. University of Victoria, March 1990.
19. G. Janssens, M. Bruynooghe, and V. Englebert. Abstracting numeric values in CLP(H,N). In *PLILP'94*, pages 400–414. Springer-Verlag, 1994.
20. M. Karr. Affine Relationships Among Variables of a Program. *Acta Informatica*, 6:133–151, 1976.
21. D. B. Kemp and P. J. Stuckey. Analysis Based Constraint Query Optimisation. In *ICLP'93*, pages 666–682, 1993.
22. A. Kerbrat. Personal Communication on Polyhedral library `polyhedra.tar`. October, 1993.
23. J. Martin, A. King, and P. Soper. Typed Norms for Typed Logic Programs. In *LOPSTR'96*. Springer-Verlag, 1996.
24. F. Mesnard and J.-G. Ganascia. CLP($\mathcal{Q}$) for Proving Interargument Relations. In *META'92*, pages 308–320, Uppsala, Sweden, 1992. Springer-Verlag.
25. H. Millroth. *Reforming Compilation of Logic Programs*. PhD thesis, Computing Science Department, Uppsala University, 1990.
26. H. Millroth. Personal Communication on the rôle of Argument Relationships in Reform. October, 1993.
27. L. Plümer. *Termination Proofs for Logic Programs*. Springer-Verlag, 1990.
28. K. Sohn. Constraints among Argument Sizes in Logic Programs. In *PODS'94*, pages 68–74. ACM Press, 1994.
29. J. D. Ullman. Implementation of Logical Query Languages for Databases. *ACM Transactions on Database Systems*, 10(3):289–321, 1985.
30. A. Van Gelder. Deriving constraints among argument sizes in logic programs. *Annals of Mathematics and Artifical Intelligence*, (3), 1991.
31. K. Verschaetse and D. De Schreye. Derivation of Linear Size Relations by Abstract Interpretation. In *PLILP'92*, pages 296–310. Springer-Verlag, 1992.
32. D. Wilde. A Library for doing Polyhedral Operations. Technical Report PI-785, Institut de Recherche en Informatique et Systemes Aleatoires, Campus Universitaire de Beauilieu, 35042 Rennes Cedex, France, 1993.

## A Selection of Example Analyses

To compare the usefulness of our implementation with previously proposed analyses [30], we list some programs which have traditionally proved difficult to analyse, complete with the results of our analysis.

`Leq/2` The `Leq(x, y)` predicate, adapted from [30, pp. 55], holds if $x$ is less or equal to $y$. Non-negative integers are encoded in successor notation. The size of `Succ`($t$) is taken to be the size of the term $t$ plus one. The `Leq(x, y)` predicate illustrates a class of predicate which for the analysis of [30] does *not* terminate. In [8] a related predicate is analysed in a finite number of iterations through widening. The widening refines those proposed in [9, 18] and is precise enough, like ours, to infer $x \leq y$. For comparison with [8], the sizes of terms are asserted to be non-negative.

`Trd/2` The affine approach [14, 20, 31] cannot deduce any information for the `Trd(x, y)` predicate of [31, pp. 308]. By way of contrast, our combination of polyhedral sets and widening can infer some interesting results. Note that the non-negativity of $x$ and $y$ follows from $\frac{2}{3}x \leq y$ and $y \leq \frac{3}{2}x$. The norm in this and the following examples is taken to be list length. Note that since the prototype analyser is implemented in CLP($\mathcal{R}$) the analysis actually outputs $0.666667x \leq y \wedge y \leq 1.5x$.

`Perm/2` The `Perm(l,p)` predicate enumerates all the permutations `p` of the list `l`. To deduce termination,it is necessary to infer that `l` in the recursive call is smaller than `[h|t]`. This can only be inferred by deducing an inter-argument relation for `Del/3` - that its third argument is smaller than the second argument. This is inferred.

`Sp/3` The invariants for the `Sp/3` predicate listed in section 1 were inferred automatically, but were re-arranged for reasons of clarity. `Sp/3` is structurally recursive but has a subtle twist in that the recursive call switches the last two arguments around. The relationships that can be deduced for the three arguments are that the second and third sum to the first and that the second will either be the same size as or, at most 1 element larger than the third.

| *Program* | *Abstract program* | *Inter-argument relationships* |
|---|---|---|
| `Leq(x,x).`<br>`Leq(x,Succ(y)) <-`<br>`   Leq(x,y).` | $\texttt{Leq}^{\mathcal{A}}\texttt{(x,x) <-}$<br>$\quad \texttt{0} \leq \texttt{x.}$<br>$\texttt{Leq}^{\mathcal{A}}\texttt{(x,1+y) <-}$<br>$\quad \texttt{0} \leq \texttt{x \&}$<br>$\quad \texttt{0} \leq \texttt{y \&}$<br>$\quad \texttt{Leq}^{\mathcal{A}}\texttt{(x,y).}$ | $\texttt{Leq}^{\mathcal{A}}(x,y)$ `<-`<br>$\quad x \leq y$ `&`<br>$\quad (-x) \leq 0$ `&`<br>$\quad (-y) \leq 0.$ |
| `Trd([],[]).`<br>`Trd([_,_\|x],[_,_,_\|y]) <-`<br>`   Trd(x,y).`<br>`Trd([_,_,_\|z],[_,_\|t]) <-`<br>`   Trd(z,t).` | $\texttt{Trd}^{\mathcal{A}}\texttt{(0,0).}$<br>$\texttt{Trd}^{\mathcal{A}}\texttt{(2+x,3+y) <-}$<br>$\quad \texttt{Trd}^{\mathcal{A}}\texttt{(x,y).}$<br>$\texttt{Trd}^{\mathcal{A}}\texttt{(3+z,2+t) <-}$<br>$\quad \texttt{Trd}^{\mathcal{A}}\texttt{(z,t).}$ | $\texttt{Trd}^{\mathcal{A}}(x,y)$ `<-`<br>$\quad \frac{2}{3}x \leq y$ `&`<br>$\quad y \leq \frac{3}{2}x.$ |
| `Perm([],[]).`<br>`Perm([h\|t],[a\|p]) <-`<br>`   Del(a,[h\|t],l) &`<br>`   Perm(l,p).`<br><br>`Del(x,[x\|y],y).`<br>`Del(u,[y\|z],[y\|w]) <-`<br>`   Del(x,z,w).` | $\texttt{Perm}^{\mathcal{A}}\texttt{(0,0).}$<br>$\texttt{Perm}^{\mathcal{A}}\texttt{(1+t,1+p) <-}$<br>$\quad \texttt{Del}^{\mathcal{A}}\texttt{(\_,1+t, l) \&}$<br>$\quad \texttt{Perm}^{\mathcal{A}}\texttt{(l,p).}$<br><br>$\texttt{Del}^{\mathcal{A}}\texttt{(\_,1+y,y).}$<br>$\texttt{Del}^{\mathcal{A}}\texttt{(\_,1+z,1+w) <-}$<br>$\quad \texttt{Del}^{\mathcal{A}}\texttt{(\_,z,w).}$ | $\texttt{Perm}^{\mathcal{A}}(x,y)$ `<-`<br>$\quad y = x$ `&`<br>$\quad (-x) \leq 0.$<br><br>$\texttt{Del}^{\mathcal{A}}(x,y,z)$ `<-`<br>$\quad z = y + 1.$ |
| `Sp([],[],[]) <-`<br>`Sp([x\|xs],[x\|os],es) <-`<br>`   Sp(xs,es,os).` | $\texttt{Sp}^{\mathcal{A}}\texttt{(0,0,0) <-}$<br>$\texttt{Sp}^{\mathcal{A}}\texttt{(1+xs,1+os,es) <-}$<br>$\quad \texttt{Sp}^{\mathcal{A}}\texttt{(xs,es,os).}$ | $\texttt{Sp}^{\mathcal{A}}(x,y,z)$ `<-`<br>$\quad z = -y + x$ `&`<br>$\quad y - x \leq 0$ `&`<br>$\quad y - \frac{1}{2}x \leq \frac{1}{2}$ `&`<br>$\quad (-y) + \frac{1}{2}x \leq 0$ `&`<br>$\quad (-y) \leq 0.$ |

This article was processed using the LaTeX macro package with LLNCS style