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([x|xs], s) <-
   Pt(x, xs, l, g) &
   Qs(l, sl) & Qs(g, sg) &
   Ap(sl, [x|sg], s).
```

```
Qs^A(0, 0).
Qs^A(1+xs, s) <-
   Pt^A(_, xs, l, g) &
   Qs^A(l, sl) & Qs^A(g, sg) &
   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([], [], []).                 Sp^A(0, 0, 0).
Sp([x|xs], [x|os], es) <-       Sp^A(1+xs, 1+os, es) <-
   Sp(xs, es, os).                 Sp^A(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).                    Ap^A(0, s, s).
Ap([x|xs], s, [x|t]) <-          Ap^A(1 + r, s, 1 + t) <-
   Ap(xs, s, t).                    Ap^A(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 `Ap`$^{\mathcal{A}}$`/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

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

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:

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

□

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 $\mathbf{N}$-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 $\triangledown : L \times L \to L$ 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, $F : L \to L$ is continuous, $\bot \in L$ is such that $\bot \sqsubseteq F(\bot)$, $\triangledown \in L \times L \to L$ is a widening, then the upward iteration sequence with widening x_i where $i, k \in \mathbf{N}$ is defined thus:

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

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 γ. Concretisation $\gamma(I) : \wp(B_{Lin}) \to \wp(B_{Herb})$ is defined by:

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

□

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:

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

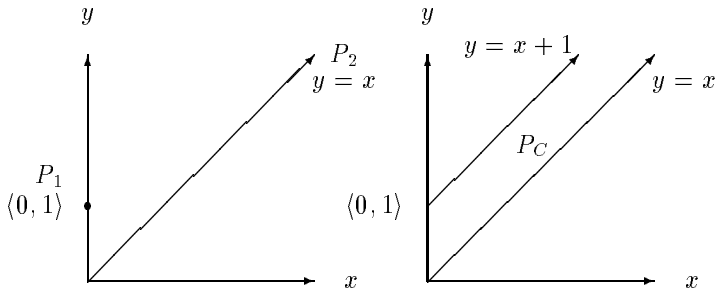
The convex hull P of two polyhedra P_1 and P_2, respectively represented by $\langle V_1, R_1 \rangle$ and $\langle V_2, R_2 \rangle$, is then given by $\langle V, R \rangle$ where $V = V_1 \cup V_2$ and $R = R_1 \cup R_2$

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.

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

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

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



□

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:

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

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:

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

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:

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

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

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

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 $\wp(B_{Lin})/\approx\ (\sqsubseteq)$, $\bot \sqsubseteq T_P(\bot)$ where $\bot = [\emptyset]_\approx$, by Proposition 3 it only remains to define a suitable $\bigtriangledown$ operator for $\wp(B_{Lin})/\approx$ and to select an appropriate k. The widening $[I]_\approx \bigtriangledown [I']_\approx$ 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.

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

$$\text{where } c \bigtriangledown c' = \{\, i \in c \;\big|\; c' \models i \,\}$$

□

The widening $c \bigtriangledown c'$ 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 $c \bigtriangledown c' = c \cap c'$ 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 $\texttt{Ap}^{\mathcal{A}}/3$ 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 $\texttt{Ap}^{\mathcal{A}}/3$ 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.