# Constraints among Argument Sizes in Logic Programs

(Extended Abstract)

**Kirack Sohn**
University of California, Santa Cruz
ksohn@cse.ucsc.edu *

## Abstract

In logic programs the argument sizes of derivable facts w.r.t. an $n$-ary predicate are viewed as a set of points in $R^n$, which are approximated by their convex hull. *Interargument constraint* w.r.t. a predicate is essentially a set of constraints that every derivable fact of the predicate satisfies. We formalize such constraints by a fixpoint of *recursive transformation* similar to immediate consequence operator. However, the transformation does not necessarily converge finitely. Approximating polycones to their affine hulls provides useful interargument constraints in many practical programs, guaranteeing finite convergence. For a class of linear recursive logic programs satisfying *translativeness* property, precise interargument constraints can be obtained by an analysis of structures of recursive transformations.

## 1 Introduction

Automated termination detection is an essential tool for generating a suitable evaluation strategy in modern deductive database systems, such as LDL and NAIL!. A deductive database is divided into two components: an *extensional* database (EDB), which consists of a set of database facts, and an *intensional* database (IDB), which consists of a set of rules defining how additional relations are computed. IDBs may be evaluated either *top-down* or *bottom-up*. Bottom-up evaluation is often said to be more efficient for logic programs with finite domains (Datalog); however, there is no corresponding claim for general logic programs with function symbols. *Capture rules* were introduced to decide which evaluation strategy is efficiently applicable to a logic program provided with a goal [Ull85, MUVG86]. A capture rule is a statement of the form: "if the rules satisfy such-and-such conditions, then a good evaluation method is such-and-such." A minimal requirement to apply top-down evaluation strategy is to guarantee termination for any query of interest.

Termination detection methods in the published papers[UVG88, APP+89, BS89b, Plü90, SVG91] are based on the relationship among argument sizes of predicates, called "interargument constraint" (for short, IC). This paper concerns how to infer such information on predicates.

**Example 1:** Consider a recursive rule:

```
p(s(X)) :- p(X).
```

Suppose the rule is called with its argument bound. In this case, termination can be shown since the argument of a (recursive) goal is a subterm of that of the previous goal and a subterm ordering over bound terms cannot have an infinitely decreasing sequence.

It is often the case that there are no direct relationships among the argument sizes in a head and those in a recursive subgoal. Consider another recursive rule:

```
p(X) :- a(X,Y), p(Y).
```

Since `X` is processed into `Y` somehow by the subgoal `a`, we cannot establish a direct relationship between `X` and `Y`. Given an ordering $>$ which has no infinitely decreasing sequences, suppose we infer all the facts derived by the predicate `a` satisfy a constraint $a_1 > a_2$ where $a_1$ and $a_2$ denote the first and second argument of `a`, resp. The constraint establishes the argument of a recursive goal are getting smaller, proving termination. Such a constraint as $a_1 > a_2$ is called an interargument constraint and the ordering we use in this paper is termsize (see Definition 1). □

We believe IC is also useful for showing safety of queries in deductive databases [BS91], and for deciding granularity of tasks in the parallel execution of logic programs [DLH90] as well as for termination analysis.

Methods to derive IC have been studied recently in terms of Datalog [BS89a, BS91] or logical rules with function symbols [VG91]. In their methods IC is

*Work partially supported by NSF grants CCR-89-58590 and IRI-9102513

formalized by a fixpoint of bottom-up inference operator similar to "immediate consequence operator". In [BS91] IC is represented in the form of a disjunctive union of inequalities between two argument sizes. Sometimes we fail to detect termination using this type of IC (for example, see Example 3.1 in [SVG91]). Van Gelder studied a method to derive IC in the form of a single polyhedral convex set [VG91]. This type of IC corresponds to relationship among many argument sizes. Both methods often fail to finitely converge. In this paper, we extend Van Gelder's work by providing two practical techniques to capture IC in finite time.

Section 2 introduces basic concepts on size abstraction and convex sets. In Section 3, we formalize "recursive transformation" whose fixpoint is IC. In Section 4, we introduce a technique called "affine widening". This technique accelerates the convergence of the recursive transformation up to finitely many iterations, yet supplying useful ICs. In Section 5, we introduce so-called "translativeness property". For linear recursive logic procedures satisfying translativeness property, we provide a technique to find precise IC (corresponding to lfp of the transformation) without iterating the transformation. Whether a procedure satisfies translativeness property can be tested by analyzing the relationship between argument sizes of the head and those of the recursive subgoal. Many practical programs relying on "recursion on structures" satisfy the translativeness property. IC which cannot be derived by other methods can be found. Section 6 concludes the paper.

## 2 Basic Concepts

Logic programs can be abstracted by the sizes of terms. Our method does not depend on any specific size definition. In examples we shall use *termsize*, that has been used for termination analysis in [VG91, SVG91].

**Definition 1:** Termsize is defined as follows:

$$termsize(t) = \begin{cases} n + \sum_{i=1}^{n} termsize(t_i) & \text{if } t = f(t_1, ..., t_n) \\ t & \text{if } t \text{ is a variable} \\ 0 & \text{if } t \text{ is a constant} \end{cases}$$

It is informally the number of edges in the tree representation of a ground term. For terms containing logical variables, a real variable $x$ constrained to nonnegativity is associated with each logical variable `X`. For instance, the structural term size of `f(a,g(X),X)` is and $4 + 2x$. □

A rule is abstracted by replacing terms by their termsizes. We shall call the resulting rules and programs *abstract rules* and *abstract programs*, resp.

**Example 2:** Consider usual `append` procedure.

```
a([],T,T).
a([X|U],V,[X|W]) :- a(U,V,W).
```

Replacing terms by their termsizes reduces to the following CLP($R$)-like procedure.

$a(0, t, t) \leftarrow t \geq 0.$
$a(2 + x + u, v, 2 + x + w) \leftarrow (x, u, v, w) \geq \vec{0}, a(u, v, w).$

□

The $i$-th argument size of predicate $p$ is denoted by $p_i$ and $\vec{p}$ denotes a vector $(p_1, \ldots, p_n)$ for $n$-ary predicate `p`; for example, $a_1 = 0, a_2 = t, a_3 = t$ or $\vec{a} = (0, t, t)$ for the base case rule of Example 2.

A *polyhedral convex set* in $R^n$ is the intersection of finitely many closed half-spaces. A *polycone* is a polyhedral convex set in the nonnegative orthant of $R^n$. IC with respect to an $n$-ary predicate `p` is captured in the form of a polycone in $R^n$. A polycone in $R^n$ is usually represented by an affine image of another polycone in $R^m$; that is,

$$\{\vec{p} \mid \exists \vec{x}.[(\vec{p} = \vec{a} + A\vec{x}) \wedge C(\vec{x})]\} \qquad (1)$$

where $\vec{p}$ is the vector of variables representing argument sizes, $\vec{x}$ is the vector of *parameters*, $\vec{a}$ is a vector constant, $A$ is a matrix, and $C(\vec{x})$ is a set of constraints in parameter space, including nonnegativity constraints for parameters. Eq. 1 is called the *parametric representation* of a polycone. An empty polycone is denoted by $\emptyset$.

The parametric representation of a polycone can be generated by a tuple of a set of points and a set of rays called a *generator*. The polycone is a vector sum of a convex hull of points $X_1, \ldots, X_n$ and a cone of rays $Y_1, \ldots, Y_m$; that is,

$$\left\{ \vec{p} \;\middle|\; \exists \vec{\lambda}. \left[ \begin{array}{ll} & \left( \begin{array}{ll} \vec{p} = & \lambda_1 X_1 \\ & \vdots \\ + & \lambda_n X_n \\ + & \lambda_{n+1} Y_1 \\ & \vdots \\ + & \lambda_{n+m} Y_m \end{array} \right) \\ \wedge & (\sum_{i=1}^{n} \lambda_i = 1) \\ \wedge & (\lambda_i \geq 0, i = 1, \ldots, n+m) \end{array} \right] \right\}. \qquad (2)$$

We shall say that a polycone in Eq. 2 is *generated* by points $X_1, \ldots, X_n$ and rays $Y_1, \ldots, Y_m$. A unique minimal generator called a *frame* is obtained by eliminating nonextreme points and rays from the generator. Equivalence of two polycones can be tested by comparing their frames. A generator and a frame of a polycone $\Delta$ are denoted by gen($\Delta$) and frame($\Delta$), resp. For example, let $\Delta$ be a polycone corresponding to Eq. 2, gen($\Delta$) is $(\{X_1, \ldots, X_n\}, \{Y_1, \ldots, Y_m\})$.

The *convex union* $\Delta_1 \sqcup \Delta_2$ of two polycones $\Delta_1$ and $\Delta_2$ is defined as the closure of the set of points that are convex combinations of any two points of union of two polycones. $\sqcup$ is commutative and associative. A convex union of $n$ polycones $\Delta_1, \ldots, \Delta_n$ is denoted by $\sqcup\{\Delta_1, \ldots, \Delta_n\}$. The generator of $\sqcup\{\Delta_1, \ldots, \Delta_n\}$ is a tuple of the union of sets of points and the union of sets of rays in the generators of $\Delta_i$'s.

The *affine hull* aff($\Delta$) of a polycone $\Delta$ is the smallest subspace containing $\Delta$. If

$$\text{gen}(\Delta) = (\{X_1, \ldots, X_n\}, \{Y_1, \ldots, Y_m\}),$$

then

$$\text{aff}(\Delta) = \left\{ \vec{p} \;\middle|\; \exists \vec{\lambda}. \left[ \begin{array}{ll} & \left( \begin{array}{ll} \vec{p} = & \lambda_1 X_1 \\ & \vdots \\ + & \lambda_n X_n \\ + & \lambda_{n+1} Y_1 \\ & \vdots \\ + & \lambda_{n+m} Y_m \end{array} \right) \\ \wedge & (\sum_{i=1}^{n} \lambda_i = 1) \end{array} \right] \right\}. \quad (3)$$

See [Sch86, Roc70] for details on polyhedral convex sets.

A predicate dependency graph is a digraph with nodes of predicates and arcs from $p$ to $q$ where $p$ is a head predicate of a certain rule and $q$ is its subgoal predicate. Intuitively, $q$ supports the derivation, or solution, of $p$. We identify strongly connected components (SCC) of this digraph, and a directed acyclic graph (DAG) whose nodes are SCCs. In the actual derivation of ICs, we analyze one SCC at a time rather than the whole program. The analysis of an SCC is supported by ICs from its lower SCCs.

## 3 Recursive Transformations

Suppose there are $k$ predicates $p^1, \ldots, p^k$ in a program $P$, and let $a(p^i)$ denote the arity of $p^i$. Let $\Delta_{p^1}, \ldots, \Delta_{p^k}$ be polycones in $R^{a(p^1)}, \ldots, R^{a(p^k)}$ resp., and $\Delta = (\Delta_{p^1}, \ldots, \Delta_{p^k})$. Let $\mathcal{U}_{p^i}$ be the set of all convex sets in the positive orthant of $R^{a(p^i)}$ and $\mathcal{U} = \mathcal{U}_{p^1} \times \cdots \times \mathcal{U}_{p^k}$. $\mathcal{U}$ forms a complete lattice equipped with componentwise subset ordering $\sqsubseteq$. We now introduce a transformation corresponding to one logical rule.

**Definition 2:** (natural transformation) Suppose the $i$-th non-base abstract rule whose head predicate is $p$ is in the form of:

$$p(\vec{e}) \leftarrow \vec{\lambda} \geq \vec{0}, q(\vec{f}), \ldots r(\vec{g}). \quad (4)$$

where $\vec{\lambda}$ is a vector of variables appearing in the rule. $\vec{e} = (e_1, \ldots, e_{a(p)})$, $\vec{f} = (f_1, \ldots, f_{a(q)})$, $\ldots$, $\vec{g} = (g_1, \ldots, g_{a(r)})$ are vectors of linear arithmetic terms. The *natural transformation* $\Psi_{<p,i>} : \mathcal{U} \rightarrow \mathcal{U}_p$ corresponding to the $i$-th rule of predicate $p$ is defined by:

$$\Psi_{<p,i>}(\Delta) = \left\{ \vec{p} \;\middle|\; \exists \vec{\lambda}. \left[ \begin{array}{ll} & (\vec{p} = \vec{e}) \\ \wedge & (\vec{\lambda} \geq \vec{0}) \\ \wedge & (\vec{f} \in \Delta_q) \\ & \vdots \\ \wedge & (\vec{g} \in \Delta_r) \end{array} \right] \right\}. \quad (5)$$

For $i$-th base abstract rule in the form of:

$$p(\vec{e}) \leftarrow \vec{\lambda} \geq \vec{0}. \quad (6)$$

we have the following *base* polycone:

$$B_{<p,i>} = \{\vec{p} \mid \exists \vec{\lambda}.[(\vec{p} = \vec{e}) \wedge (\vec{\lambda} \geq \vec{0})]\}. \quad (7)$$

□

**Example 3:** Continuing with Example 2, suppose we have a parametric representation of a polycone:

$$\Delta = \{\vec{a} \mid \exists t.[(\vec{a} = (0, t, t)) \wedge (t \geq 0)]\}.$$

Then

$$\Psi_{<a,1>}(\Delta) = \left\{ \vec{a} \;\middle|\; \exists x u v w t. \left[ \begin{array}{ll} & ((x, u, v, w, t) \geq \vec{0}) \\ \wedge & (a_1 = 2 + x + u) \\ \wedge & (a_2 = v) \\ \wedge & (a_3 = 2 + x + w) \\ \wedge & (u = 0) \\ \wedge & (v = t) \\ \wedge & (w = t) \end{array} \right] \right\}.$$

For the base rule, we have a polycone: $B_{<a,1>} = \{\vec{a} \mid \exists t.[(\vec{a} = (0, t, t)) \wedge (t \geq 0)]\}$. □

We extend this formalism to the whole program $P$ in a natural way.

**Definition 3:** (recursive transformation) A *recursive transformation* $T_P : \mathcal{U} \rightarrow \mathcal{U}$ associated with a program $P$ is a direct product of $T_{p^1}, \ldots, T_{p^k}$. $T_{p^i}$ is defined by a convex union of $l$ base polycones and $m$ natural transformations associated with the rules whose head predicate is $p^i$.

$$T_P(\Delta) = (T_{p^1}(\Delta), \ldots, T_{p^k}(\Delta))$$
$$T_{p^i}(\Delta) = \sqcup \left\{ \begin{array}{l} B_{<p^i,1>}, \ldots, B_{<p^i,l>}, \\ \Psi_{<p^i,1>}(\Delta), \ldots, \Psi_{<p^i,m>}(\Delta) \end{array} \right\} \quad (8)$$

□

To maintain a single polycone for each predicate p, we take the convex union of polycones from all the rules whose head predicate is p. Natural transformation and recursive transformation are similar to those in [VG91], but more general.

**Theorem 1:** A recursive transformation $T_P$ is monotone. □

**Theorem 2:** There exists the least fixpoint associated with $T_P$. □

Our formalism is an instance of abstract interpretation, hence correctness is guaranteed. A fixpoint can be verified by comparing the frames of polycones in $\Delta$ and $T(\Delta)$ componentwise.

**Example 4:** Continuing with Example 2, we now compute the least fixpoint of recursive transformation. Since there is only one predicate `a` in a program, Let $B, \Psi, T$ denote $B_{<a,1>}, \Psi_{<a,1>}, T_P = T_a$, resp. Note that $\emptyset$ denotes an empty polycone. How to find extreme points and rays can be found in [VG91].

1. (base polycone) $B = \{\vec{a} \mid \exists t.[(t \geq 0) \wedge (\vec{a} = (0, t, t))]\}$ and frame$(B) = (\{(0,0,0)\}, \{(0,1,1)\})$
2. (natural transformation) $\Psi(\emptyset) = \emptyset$
3. (recursive transformation; convex union of $B$ and $\emptyset$) frame$(T(\emptyset)) = (\{(0,0,0)\}, \{(0,1,1)\})$
4. (verify a fixpoint) frame$(\emptyset) \neq$ frame$(T(\emptyset))$, so generate the parametric representation from the frame of $T(\emptyset)$
5. (natural transformation) $\Psi(T(\emptyset)) = \{\vec{a} \mid \exists xuvwt. [((x,u,v,w,t) \geq \vec{0}) \wedge (\vec{a} = (2+x+u, v, 2+x+w)) \wedge (u = 0) \wedge (v = t) \wedge (w = t)]\}$, and frame$(\Psi(T(\emptyset))) = (\{(0,0,0)\}, \{(0,1,1),(1,0,1)\})$
6. (recursive transformation; convex union of $B$ and $\Psi(T(\emptyset))$) frame$(T^2(\emptyset)) = (\{(0,0,0)\}, \{(0,1,1),(1,0,1)\})$
7. (verify a fixpoint) frame$(T^1(\emptyset)) \neq$ frame$(T^2(\emptyset))$, so generate the parametric representation from the frame of $T^2(\emptyset)$

With one more recursive transformation, we reach the least fixpoint. IC generated by the frame of lfp is $\{\vec{a} | \exists uv.[((u,v) \geq \vec{0}) \wedge (\vec{a} = (u, v, u+v))]\}$. In `append` procedure, its third argument size is equal to the sum of its first and second. □

## 4 Affine Widening

Unfortunately, recursive transformations associated with many practical programs often fail to converge finitely.

**Example 5:** Consider an abstract procedure below:

$p(t,t) \leftarrow t \geq 0.$
$p(u, 2+v) \leftarrow (u,v) \geq \vec{0}, p(u,v).$

Since the second (recursive) rule increases the input by (0,2) at every transformation, we reaches the least fixpoint $\{(p_1,p_2) \mid \exists uv.[(u \geq 0) \wedge (v \geq 0) \wedge (p_1 = u) \wedge (p_2 = u+v)]\}$ after infinite recursive transformations. □

Whether a recursive transformation converges finitely is believed to be undecidable[1]. In logic programs data structures in input arguments are usually transferred to output arguments even though they may be broken into smaller ones, or combined into larger ones, or the values of elements are processed. In terms of argument sizes, the relationship among them may be approximated by an affine subspace, which is represented by a set of equalities, Requiring the least fixpoint is too much in general; however, a postfixpoint [2] is also a correct upper approximation to least fixpoint by Tarski's fixpoint theorem: $T(x) \leq x$ implies lfp$T \leq x$. The idea of using widening has already been used in the abstract interpretation of programs in procedural languages [CH78].

We now describe what we call *affine widening*. To simplify notations, let us omit subscripts concerning predicate names and suppose there is only one predicate in a program. $A_i$ is an affine subspace w.r.t. the predicate. If $A_i$ is a postfixpoint of $T$, we are done with a correct IC. Otherwise, we widen $T(A_i)$ to its affine hull $A_{i+1}$. Suppose $A_l$ is such a postfixpoint that it is found for the first time in the computation of $A_0 = \emptyset, A_1, \ldots$.

**Theorem 3:** The sequence $A_0, A_1, \ldots, A_l$ is finite.

**Proof:** It is enough to show the dimension of $A_{i+1}$ is greater than that of $A_i$ for $i = 0, \ldots, l-1$ since $A_i$'s are affine subspaces in $R^n$ where $n$ is the arity of a predicate associated with the recursive transformation $T$. $A_i \subset T(A_i)$ since $A_i$ is not a postfixpoint of $T$ and $T(A_i) \subseteq A_{i+1}$ since $A_{i+1}$ is an affine hull of $T(A_i)$. So $A_{i+1}$ is an affine subspace properly including $A_i$; that is, the dimension of $A_{i+1}$ is greater than that of $A_i$. □

Although affine widening sometimes widens to "no constraints", it provides useful IC with many practical programs.

**Example 6:** Let us consider `append` procedure with a constraint that all list elements are integers.

```
a([],T,T).
a([X|U],V,[X|W]) :- integer(X), a(U,V,W).
```

The purpose of introducing the constraint `integer(X)` is to set the termsize of `X` to zero. The abstracted version of the above `append` procedure is as follows:

[1] Brodsky and Sagiv studied inference of disjunction of inequalities between two argument positions. With a transformation similar to our recursive transformation, they proved the question on convergence is undecidable[BS91]

[2] If $x \geq T(x)$ ($x \leq T(x)$), then $x$ is a postfixpoint (prefixpoint) of $T$.

$a(0,t,t) \leftarrow t \geq 0.$
$a(2+x+u,v,2+x+w) \leftarrow$
$\quad (x,u,v,w) \geq \vec{0},$
$\quad x = 0,$
$\quad a(u,v,w).$

Using recursive transformation without affine widening, the transformation does not converge finitely since it expands the input convex set inch by inch at every iteration. We now try with affine widening. $A_1 = T(\emptyset) = \{(a_1,a_2,a_3) \mid a_1 = 0, a_2 = a_3\}$ is an affine subspace, which is one-dimensional. frame$(T(A_1)) = (\{(0,0,0),(2,0,2)\},\{(0,1,1)\})$, so its affine hull is $A_2 = \{(a_1,a_2,a_3) \mid a_3 = a_1 + a_2\}$, which is two-dimensional. With one more iteration, we find that $A_2$ is a post-fixpoint of $T$ (fortunately, a fixpoint). Van Gelder's method does not finitely converge [VG91]. Ullman and Van Gelder's [UVG88] and Brodsky and Sagiv's methods [BS91] cannot represent a relationship among three argument positions. However, it is noted that affine widening cannot infer relationship in the form of inequality. □

## 5 Translativeness Property

Let $\vec{p}$ be a vector of argument sizes in a head and $\vec{p'}$ a vector of argument sizes in a recursive subgoal. All nonrecursive subgoals are replaced by their ICs. Then an abstract rule can be viewed as a polycone in $R^{2n}$, so the polycone can be re-generated by its extreme points and rays.

**Example 7:** Consider the naive `reverse` procedure.

```
r([],[]).
r([X|U],W) :- r(U,V), a(V,[X],W).
```

Applying termsize abstraction to the procedure, we have an abstract procedure:

$r(0,0).$
$r(2+x+u,w) \leftarrow (u,v,w) \geq \vec{0},\ r(u,v),\ a(v,2+x,w).$

The analysis for `a` supplies an IC: $a_1 + a_2 = a_3$. Replacing $a(v,2+x,w)$ by $v+(2+x) = w$ reduces to the following abstract procedure.

$r(0,0).$
$r(2+x+u,w) \leftarrow (u,v,w) \geq \vec{0},\ r(u,v),\ v+2+x = w.$

Let $\vec{r}$ and $\vec{r'}$ denote the argument sizes of head and recursive subgoal, resp. So we have the following parametric representation for $r_1, r_2, {r_1}', {r_2}'$.

$$\begin{array}{l} r_1 = 2+x+u, \\ r_2 = w, \\ {r_1}' = u, \\ {r_2}' = v, \\ v+2+x = w, \\ x \geq 0, \\ u \geq 0, \\ v \geq 0, \\ w \geq 0 \end{array}$$

The parametric representation generated by the extreme points and rays is:

$$\begin{array}{lcllll} r_1 & = & 2 & +x & +u & \\ r_2 & = & 2 & +x & & +v \\ {r_1}' & = & 0 & & +u & \\ {r_2}' & = & 0 & & & +v \\ x & \geq & 0 & & & \\ u & \geq & 0 & & & \\ v & \geq & 0 & & & \end{array}$$

□

A natural transformation $\Psi$ associated with a linear recursive rule is viewed as a polycone in $R^{2n}$ when all nonrecursive subgoals are replaced by their ICs. Suppose the parametric representation generated by extreme points and rays of the polycone in $R^{2n}$ is in the form of :

$$\Psi = \left\{ (\vec{p},\vec{p'}) \;\middle|\; \exists \vec{\lambda}\vec{\mu}\vec{\rho}. \left[\begin{array}{ll} & (\text{Eq. 10}) \\ \wedge & (\sum \lambda_i = 1) \\ \wedge & (\vec{\lambda} \geq \vec{0}) \\ \wedge & (\vec{\mu} \geq \vec{0}) \\ \wedge & (\vec{\rho} \geq \vec{0}) \end{array}\right] \right\}. \quad (9)$$

$$\begin{pmatrix} \vec{p} \\ \vec{p'} \end{pmatrix} = \begin{pmatrix} A \\ 0 \end{pmatrix} \vec{\lambda} + \begin{pmatrix} B \\ 0 \end{pmatrix} \vec{\mu} + \begin{pmatrix} P_1 \\ P_2 \end{pmatrix} \vec{\rho} \quad (10)$$

where $P_1$ and $P_2$ are permutations. Substituting $\rho = P_2^{-1}\vec{p'}$ in Eq. 10 boils down to:

$$\Psi = \left\{ \vec{p} \;\middle|\; \exists \vec{\lambda}\vec{\mu}. \left[\begin{array}{ll} & (\vec{p} = A\vec{\lambda} + B\vec{\mu} + P\vec{p'}) \\ \wedge & (\sum \lambda_i = 1) \\ \wedge & (\vec{\lambda} \geq \vec{0}) \\ \wedge & (\vec{\mu} \geq \vec{0}) \end{array}\right] \right\} \quad (11)$$

where $P$ is $P_1 P_2^{-1}$.

**Definition 4:** (translativeness) If a natural transformation $\Psi$ can be reduced to Eq. 11, $\Psi$ is *$P$-translative*. Let $\Delta_0$ denote a polycone represented by:

$$\Delta_0 = \left\{ \vec{p} \;\middle|\; \exists \vec{\lambda}\vec{\mu}. \left[\begin{array}{ll} & (\vec{p} = A\vec{\lambda} + B\vec{\mu}) \\ \wedge & (\sum \lambda_i = 1) \\ \wedge & (\vec{\lambda} \geq \vec{0}) \\ \wedge & (\vec{\mu} \geq \vec{0}) \end{array}\right] \right\}. \quad (12)$$

$\Psi(\Delta)$ is indeed a vector sum + of $\Delta_0$ and $P\Delta$:

$$\Psi(\Delta) = \Delta_0 + P\Delta$$

□

Now we introduce some useful properties on polycones.

**Lemma 1:** The following equalities on polycones hold.

1. $A(\Delta_1 + \Delta_2) = A\Delta_1 + A\Delta_2$
2. $A(\Delta_1 \sqcup \Delta_2) = A\Delta_1 \sqcup A\Delta_2$
3. $\Delta_1 + (\Delta_2 \sqcup \Delta_3) = (\Delta_1 + \Delta_2) \sqcup (\Delta_1 + \Delta_3)$

□

Suppose we have $k$ base polycones and $l$ $P$-translative natural transformations. Then we can simplify the recursive transformation $T$ as follows:

$$\begin{array}{ll} & T(\Delta) \\ = & B_1 \sqcup \cdots \sqcup B_k \sqcup (\Delta_1 + P\Delta) \sqcup \cdots \sqcup (\Delta_l + P\Delta) \\ = & (B_1 \sqcup \cdots \sqcup B_k) \sqcup ((\Delta_1 \sqcup \cdots \sqcup \Delta_l) + P\Delta) \\ = & \Delta_B \sqcup (\Delta_R + P\Delta) \end{array}$$

where $\Delta_B = B_1 \sqcup \cdots \sqcup B_k$ and $\Delta_R = \Delta_1 \sqcup \cdots \sqcup \Delta_l$ .

We now describe how $P$-translative natural transformation can be unfolded to $I$-translative transformation using unfolding, where $I$ is an identity matrix.

**Definition 5:** (Unfolding) Suppose $T$ is $P$-translative transformation. $T^2(\Delta)$ is obtained by applying $T$ to $T(\Delta)$.

$$\begin{array}{ll} & T(T(\Delta)) \\ = & \Delta_B \sqcup (\Delta_R + P(\Delta_B \sqcup (\Delta_R + P\Delta))) \\ = & \Delta_{B'} \sqcup (\Delta_{R'} + P^2\Delta) \end{array}$$

where $\Delta_{B'} = \Delta_B \sqcup (\Delta_R + P\Delta_B)$ and $\Delta_{R'} = \Delta_R + P\Delta_R$.
□

Applying at most $n$ unfoldings where $n$ is the arity of the associated predicate, we can get $I$-translative transformation since $P$ is a permutation matrix.

**Theorem 4:** Let $T$ be a translative transformation in the form of:

$$T(\Delta) = \Delta_B \sqcup (\Delta_R + \Delta)$$

and the generators of $\Delta_B$ and $\Delta_R$ are:

$$\begin{array}{l} \mathrm{gen}(\Delta_B) = (\{u_1, \ldots, u_k\}, \{v_1, \ldots, v_l\}) \\ \mathrm{gen}(\Delta_R) = (\{x_1, \ldots, x_n\}, \{y_1, \ldots, y_m\}) \end{array}$$

then

$$\mathrm{gen}(\mathsf{lfp}T) =$$
$$(\{u_1, \ldots, u_k\}, \{v_1, \ldots, v_l, x_1, \ldots, x_n, y_1, \ldots, y_m\})$$

□

In practice, linearly recursive logic programs relying on "recursion on structures" technique usually satisfy translativeness property. Hence their tight interargument constraints can be found without any iterations.

**Example 8:** The following procedure is intended to divide its first argument into its second and third argument. In order to assure balanced division of the "input" list, the last two arguments are interchanged upon recursion.

```
d([],[],[]).
d([X|U],[X|V],W) :- integer(X), d(U,W,V).
```

Applying termsize abstraction to the procedure, we have:

$d(0,0,0)$.
$d(2+u, 2+v, w) \leftarrow (u,v,w) \geq \vec{0},\ d(u,w,v)$.

Clearly, $\mathrm{frame}(\Delta_B) = (\{0,0,0\},\ \{\})$. From the second rule, we get a $P$-translative transformation:

$$\Psi(\Delta) = \Delta_R + P\Delta$$

where

$$\Delta_R = \{(2,2,0)\}$$
$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Applying an unfolding boils down to $I$-translative transformation:

$$T^2(\Delta) = \Delta_{B'} \sqcup (\Delta_{R'} + \Delta)$$

where

$$\mathrm{frame}(\Delta_{B'}) = (\{(0,0,0),(2,2,0)\},\{\})$$

and

$$\mathrm{frame}(\Delta_{R'}) = (\{(4,2,2)\},\{\}).$$

By Theorem 4,

$$\mathsf{lfp}T = \left\{ \vec{d} \;\middle|\; \exists\lambda_1\lambda_2\lambda_3. \begin{bmatrix} & (d_1 = 2\lambda_2 + 4\lambda_3) \\ \wedge & (d_2 = 2\lambda_2 + 2\lambda_3) \\ \wedge & (d_3 = 2\lambda_3) \\ \wedge & (\lambda_1 + \lambda_2 = 1) \\ \wedge & (\lambda_1 \geq 0) \\ \wedge & (\lambda_2 \geq 0) \\ \wedge & (\lambda_3 \geq 0) \end{bmatrix} \right\}.$$

It implies that the first argument size is the sum of the second and the third, and the first argument size is strictly greater than the second or the third if the first is greater than or equal to 4 (at least two list elements), which is important information when handling termination proofs. It is noted that Van Gelder's recursive transformation does not converge

with this example and his heuristic does not work [VG91]. Sagiv and Brodsky's method can infer only inequalities between two argument positions [BS91]. Recursive transformation with affine widening described in the previous section gives an IC: $\{(d_1, d_2, d_3) \mid d_1 = d_2 + d_3\}$, which is less precise than the above. □

## 6 Conclusion

We formalized a method of deriving constraints among argument sizes in the form of polyhedral convex set. We provided an affine widening operation to guarantee finite convergence of recursive transformation, yet providing useful interargument constraints.

We defined a class of linear recursive logic programs in terms of translativeness property. For such a class, tight interargument constraints are automatically given via the analysis of the structure of transformations. In practice, most of linear recursive logic programs relying on "recursion on structure" technique satisfy translativeness property.

Our method is fully amenable to automation. We implemented the method in Prolog.

## References


[APP+89] F. Afrati, C. Papadimitriou, G. Papageorgiou, A. R. Roussou, Y. Sagiv, and J. D. Ullman. On the convergence of query evaluation. *Journal of Computer and System Sciences*, 38(2):341–359, 1989.

[BS89a] A. Brodsky and Y. Sagiv. Inference of monotonicity constraints in Datalog programs. In *Eighth ACM Symposium on Principles of Database Systems*, pages 190–199, 1989.

[BS89b] A. Brodsky and Y. Sagiv. On termination of Datalog programs. In *First International Conference on Deductive and Object-Oriented Databases*, pages 95–112, Kyoto, Japan, 1989.

[BS91] A. Brodsky and Y. Sagiv. Inference of inequality constraints in logic programs. In *Tenth ACM Symposium on Principles of Database Systems*, 1991.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the 5th ACM Symposium on Principles of Programming Languages*, pages 84–96, 1978.

[DLH90] S. K. Debray, N.-W. Lin, and M. Hermenegildo. Task granularity analysis in logic programs. In *SIGPLAN'90 Conference on Programming Language Design and Implementation*, 1990.

[MUVG86] K. Morris, J. D. Ullman, and A. Van Gelder. Design overview of the Nail! system. In *Third Int'l Conf. on Logic Programming*, pages 554–568, 1986.

[Plü90] L. Plümer. *Termination Proofs for Logic Programs*, volume 446 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1990.

[Roc70] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, NJ, 1970.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, New York, 1986.

[SVG91] K. Sohn and A. Van Gelder. Termination detection in logic programs using argument sizes. In *Tenth ACM Symposium on Principles of Database Systems*, 1991.

[Ull85] J. D. Ullman. Implementation of logical query languages for databases. *ACM Transactions on Database Systems*, 10(3):289–321, 1985.

[UVG88] J. D. Ullman and A. Van Gelder. Efficient tests for top-down termination of logical rules. *Journal of the ACM*, 35(2):345–373, 1988.

[VG91] A. Van Gelder. Deriving constraints among argument sizes in logic programs. *Annals of Mathematics and Artificial Intelligence*, 3, 1991. Extended abstract appears in Ninth ACM Symposium on Principles of Database Systems, 1990.