Higher-Precision Groundness Analysis

Michael Codish[1], Samir Genaim[1], Harald Søndergaard[2] and Peter J. Stuckey[2]

[1] Dept. of Computer Science, Ben-Gurion Univ. of the Negev, Beer-Sheva, Israel
[2] Dept. of Comp. Science and Software Eng., Univ. of Melbourne, Australia

Abstract. Groundness analysis of logic programs using Pos-based abstract interpretation is one of the clear success stories of the last decade in the area of logic program analysis. In this work we identify two problems with the Pos domain, the multiplicity and sign problems, that arise independently in groundness and uniqueness analysis. We describe how these problems can be solved using an analysis based on a domain Size for inferring term size relations. However this solution has its own shortcomings because it involves a widening operator which leads to a loss of Pos information. Inspired by Pos, Size and the LSign domain for abstract linear arithmetic constraints we introduce a new domain LPos, and show how it can be used for groundness and uniqueness analysis. The idea is to use the sign information of LSign to improve the widening of Size so that it does not lose Pos information. We prove that the resulting analyses using LPos are uniformly more precise than those using Pos.

1 Introduction

Groundness analysis of logic programs using Pos-based [7, 18, 19] abstract interpretation [9] is one of the success stories of the last decade in the area of logic program analysis. Moreover, Pos can be the basis for many other applications such as finiteness analysis, rigidity analysis, type analysis, and suspension analysis (for concurrent logic programs). It has also been applied in the context of constraint logic programs to determine when variables have obtained a final unique value [2]. Here we shall refer to such an analysis as *uniqueness* analysis.

The abstract domain Pos [7, 18] consists of the positive Boolean functions, ordered by logical consequence. For groundness analysis, the idea is to express groundness information about a set of substitutions as a Boolean function. For example, the logical reading of the formula $\psi = x \wedge (y \rightarrow z)$ for groundness analysis is: on application of any substitution, x is ground and if y becomes ground then also z does. More formally, we say that a Boolean function ψ describes a substitution θ if any set of variables that might become ground by further instantiating θ is a model of ψ. For uniqueness analysis, the situation is similar, but the Boolean function is interpreted as a statement about uniqueness dependencies. For example, if any two of the three variables in the linear constraint $x = 2y + 3z$ have a unique value then so does the third. This is described by $((x \wedge y) \rightarrow z) \wedge ((y \wedge z) \rightarrow x) \wedge ((z \wedge x) \rightarrow y)$.

Pos is attractive because its structure and corresponding abstract operations are simple and Pos-based analysers are efficient in practice [12, 22]. But the domain has its limits when we consider precision of analysis. This paper exposes two shortcomings and illustrates their effect on groundness and uniqueness analyses for logic and constraint logic programs. The first problem relates to the fact that a Boolean function does not capture information about multiplicity of variables, so the term equations $[x|xs] = [x|ys]$ and $[x|xs] = [x, x|ys]$ are both approximated by $(x \wedge xs) \leftrightarrow (x \wedge ys)$. The second problem arises because Pos abstractions for arithmetic constraints ignore the signs of coefficients, so the systems of constraints $(x - y - z = 0) \wedge (x - y = 3)$ and $(x - y - z = 0) \wedge (x + y = 3)$ are both given the Pos description $(x \leftrightarrow y) \wedge (x \rightarrow z)$.

We look at several possible solutions to the two problems. For groundness analysis, the use of size relations [3, 11, 23] addresses the multiplicity problem, and can provide groundness information. A substitution is approximated by a constraint on term sizes. For example, the inequality $x \leq y + z$ describes a substitution for which the size of the term x is bound to is smaller than or equal to the sum of the sizes of the terms y and z are bound to (for some appropriately chosen measure of term size). Groundness dependencies are captured, since any substitution which makes y and z ground must make x ground to satisfy the size constraint. While the abstract domain Size of size constraints can be shown to be at least as precise as Pos, it has infinite ascending chains and so must be applied in combination with a widening operator [11]. Widening sometimes results in the loss of groundness information so that in many cases a Pos based groundness analysis is more precise than a size relations based analysis.

For uniqueness analysis, the abstract domain LSign [20], where linear constraints are abstracted by replacing coefficients by their signs, can provide the needed information about signs of coefficients. Intuitively an LSign description $\oplus x + \ominus y + \ominus z = 0 \wedge \oplus x + \oplus y = \oplus$ defines constraints C equivalent to $a_1 x - a_2 y - a_3 z = 0 \wedge a_4 x + a_5 y = a_6,\ a_i > 0, 1 \leq i \leq 6$, for example $(x - y - z = 0) \wedge (x + y = 3)$. The coefficient information is enough to determine the uniqueness information $(x \leftrightarrow y) \wedge (x \leftrightarrow z)$. Unfortunately, LSign is aimed at capturing a different kind of information, namely whether a represented constraint must be satisfiable or not, and LSign operations quickly lose information important to uniqueness analysis.

Inspired by Size and LSign we propose a new domain, LPos, which improves on Pos, both for groundness analysis and for uniqueness analysis. The idea is to use the sign information of LSign to improve the widening of Size so that it does not lose Pos information. LPos uses the same style of description as LSign and, importantly, shares the use of an abstract Fourier elimination algorithm for the abstract projection operation. It differs crucially from LSign by using operations which maintain better uniqueness information.

2 Problems with Pos

Recall that a Boolean function is *positive* if it is satisfied by the assignment of *true* to all of its variables. The abstract domain Pos consists of the set of positive Boolean functions (together with *false* as a bottom element) ordered

```
bf(Tree, List):-
   bfq(s(zero),[Tree|QT],QT List).

bfq(zero,Q,Q,[]).
bfq(s(N),[nil|QH], QT, Es):-
   bfq(N,QH, QT, Es).
bfq(s(N),[t(L,E,R)|QH],[L,R|QT],[E|Es]):-
   bfq(s(s(N)), QH, QT, Es).
```

$$
\begin{aligned}
&\mathsf{bf}(u,v) = \\
&\quad \exists qt : \mathsf{bfq}(true, u \wedge qt, qt, v) \\
&\mathsf{bfq}(w,x,y,z) = \\
&\quad [w \wedge (x \leftrightarrow y) \wedge z] \\
&\ \vee\ [\exists n, qh : (w \leftrightarrow n) \wedge (x \leftrightarrow qh) \wedge \\
&\quad \mathsf{bfq}(n, qh, y, z)] \\
&\ \vee\ [\exists n, l, e, r, qt, qh, es : \\
&\quad (w \leftrightarrow n) \wedge (x \leftrightarrow l \wedge e \wedge r \wedge qh) \wedge \\
&\quad (y \leftrightarrow l \wedge r \wedge qt) \wedge (z \leftrightarrow e \wedge es) \wedge \\
&\quad \mathsf{bfq}(n, qh, qt, es)]
\end{aligned}
$$

Fig. 1. Breadth-first traversal of binary trees (left) with Pos analysis (right).

by logical consequence [18, 7, 19]. The abstract operations that arise in analyses are: Boolean conjunction, as a greatest lower bound, Boolean disjunction, as a least upper bound, and existential quantification, for projection. There is a substantial body of literature on this type of analysis, for example, [1, 4, 8]. In this section we describe two problems with the Pos domain.

The multiplicity problem: The first problem arises in groundness analysis of logic programs. For groundness analysis, a Boolean function ψ describes a substitution θ if any set of variables that might become ground by further instantiating θ is a model of ψ. The unification of two terms t and s is described in Pos by the Boolean formula $(x_1 \wedge x_2 \wedge \ldots \wedge x_n) \leftrightarrow (y_1 \wedge y_2 \wedge \ldots \wedge y_m)$ where $\{x_1, x_2, \ldots x_n\}$ and $\{y_1, y_2, \ldots y_m\}$ are the sets of variables in t and s.

The multiplicity problem for Pos is a loss of groundness information because the abstraction function (of a Herbrand constraint) ignores the multiplicity of variables in terms. For example, the two term equations $[x|xs] = [x|ys]$ and $[x|xs] = [x, x|ys]$ are both described by the Boolean function $(x \wedge xs) \leftrightarrow (x \wedge ys)$. The inability of a Pos analysis to distinguish the two explains why it cannot find the (otherwise correct) description $xs \leftrightarrow ys$ for the first equation.

Of course neither equation is likely to appear in real-world programs. But a similar phenomenon *is* found, less transparently, in real programs. Consider the program given in Figure 1 (left side), for breadth-first traversal of binary trees. The program uses a queue of tree nodes which is represented in the first three arguments of `bfq/4`—the first argument represents the number of elements in the queue, and the second and third represent a difference list of the elements on the queue. This setup ensures that the program can be executed in both directions. A groundness analysis using Pos consists of translating the program into the recursive definition of a Boolean function illustrated in Figure 1 (right side) and then finding a closed form using Kleene iteration. This yields a description $w \wedge (x \leftrightarrow (y \wedge z))$ for $\mathsf{bfq}(w, x, y, z)$ indicating that in any answer to a query to bfq/4, the first argument is ground, and the second is ground if and only if the third and fourth both are. This result is precise. But, consider now the analysis of the predicate bf/2. The result for $\mathsf{bf}(u, v)$ is obtained from the result for bfq/4 as follows: $\exists qt : (u \wedge qt) \leftrightarrow (qt \wedge v)$ which is $true$. Hence the Pos-based analysis

```
mg(P,T,R,B) :- T = 1, B = (105/100)*P-R.
mg(P,T,R,B) :- T > 1, T1 = T-1, P1 = (105/100)*P-R, mg(P1,T1,R,B).
```

Fig. 2. Mortgage repayment in $\mathsf{CLP}(\mathbb{R}_{\mathsf{Lin}})$.

of $\mathsf{bf}/2$ yields no useful groundness information, even though in fact the first argument to $\mathsf{bf}/2$ is ground if and only if the second is.

The sign problem: The sign problem for Pos arises in the context of uniqueness analysis of linear arithmetic constraints [6, 2]. In this context, a Boolean function of the form $(x \wedge y) \rightarrow z$ expresses that if the values of the variables x and y become uniquely determined, then the value of the variable z will have been determined as a result. A constraint such as $x = y - 1$ is described by the Pos function $x \leftrightarrow y$ and a constraint such as $x = y + 3z - 1$ by $((x \wedge y) \rightarrow z) \wedge ((y \wedge z) \rightarrow x) \wedge ((z \wedge x) \rightarrow y)$. A problem is that the Pos abstraction of an arithmetic constraint ignores the signs of the coefficients in the arithmetic constraints. In particular, it is not possible for Pos to discover linear independence amongst constraints determined by these signs.

Figure 2 shows a $\mathsf{CLP}(\mathbb{R}_{\mathsf{Lin}})$ program to calculate mortgage repayments[1]. Uniqueness analysis using Pos begins by translating the program into a (possibly recursive) definition of a positive Boolean function, just as for groundness analysis. The difference is in the translation of the primitive constraints which are abstracted according to the following abstraction function:

$$\alpha(c) = \begin{cases} \mathit{true} & \text{if } c \text{ is inequality} \\ \nabla(\mathit{vars}(c)) & \text{if } c \text{ is an equation} \end{cases}$$

where $\nabla(S) = \bigwedge_{v \in S} ((\bigwedge_{v' \in S \setminus \{v\}} v') \rightarrow v)$. The abstract version of Figure 2 is:

$$\begin{array}{l} \mathsf{mg}(p,t,r,b) = [t \wedge (b \wedge r \rightarrow p) \wedge (r \wedge p \rightarrow b) \wedge (p \wedge b \rightarrow r)] \;\vee\; [\exists t_1, p_1 : \\ \quad (t_1 \leftrightarrow t) \wedge ((p_1 \wedge p) \rightarrow r) \wedge ((p \wedge r) \rightarrow p_1) \wedge ((r \wedge p_1) \rightarrow p) \wedge \mathsf{mg}(p_1, t_1, r, b)] \end{array}$$

A closed form for $\mathsf{mg}(p,t,r,b)$ is $t \wedge ((p \wedge r) \rightarrow b)$. This expresses that any successful query will bind `T` to a unique value. Moreover, if `P` and `R` are given unique values in a query, then `B` will be determined as a result. Again, this result is less precise than we might have hoped for. If one queries `mg` with fixed values for all of the parameters, except for `P`, then `P` will be determined as a result. This information is not expressed in the result of the Pos analysis.

The problem here is that Pos cannot identify linear dependencies between constraints in a conjunction. For example, both of the systems of linear constraints: $(x - y - z = 0) \wedge (x - y = 3)$ and $(x - y - z = 0) \wedge (x + y = 3)$ are best described by $(x \leftrightarrow y) \wedge (x \rightarrow z)$. However, the former determines z while the latter does not. In fact, the uniqueness information in the two systems is best described by $(x \leftrightarrow y) \wedge z$ and $(x \leftrightarrow y) \wedge (x \leftrightarrow z)$, respectively.

[1] The interest is fixed at 5% to avoid the presence of non-linear constraints.

3 Size Relations

An interesting solution to the multiplicity problem for Pos is obtained by applying a size relations analysis [3, 23], more commonly known in the context of termination analysis [5, 16]. The interpretation of this type of analysis is parametric to a given size function on terms, called a symbolic norm [16]. Symbolic norms ($|\cdot|$) are similar to linear norms except that variables are mapped to variables. In this approach, linear equalities and inequalities express constraints on the sizes of the terms bound to variables. For example, $z = x + y$ describes those substitutions σ, for which every instance $\sigma\tau$ satisfies $|z\sigma\tau| = |x\sigma\tau| + |y\sigma\tau|$.

Size relation analysis has many applications, and different norms support different uses. Here we are interested in an application for groundness information, so adopt a norm which assigns any ground term to the size zero, and a non-ground term to the sum of the sizes of its variables. We refer to this size function as the var-norm.

Definition 1 (Var-norm). *Given Herbrand term t, we define its* var-norm *by:*

$$|t|_{vn} = \begin{cases} 0 & \textit{if } t \textit{ is a constant} \\ t & \textit{if } t \textit{ is a variable} \\ |t_1|_{vn} + \cdots + |t_n|_{vn} & \textit{if } t \textit{ has form } f(t_1, \ldots, t_n) \end{cases}$$

A size relation analysis for a logic program P can be formalised in terms of the semantics of a corresponding CLP($\mathbb{N}$) program on the sizes of the terms in P. CLP($\mathbb{N}$) stands for constraint logic programs over a domain consisting of the non-negative integers. The semantic objects are sets of CLP($\mathbb{N}$) facts modulo an equivalence relation defined in terms of constraint equivalence. The meaning of a CLP($\mathbb{N}$) program is a potentially infinite set of CLP($\mathbb{N}$) facts. In practice, analyses apply a convex hull operation as an upper bound operator to improve efficiency, and a widening operator [11] (which detects stable edges in corresponding polyhedral representations of the constraints on the term sizes) to guarantee the termination of the analysis.

There is a pleasing similarity between a size relations analysis and a groundness dependency analysis. Both fit into the framework of Giacobazzi et al. [15]. We can translate a (possibly recursive) definition of a predicate into a (possibly recursive) definition of a constraint, expressing the term size relations for the predicate. A term equation $t = s$ is translated into $|t| = |s|$. For the var-norm, the bfq program in Figure 1 is translated to:

$$\begin{aligned} \mathsf{bf}(u, v) = \;& \exists qt : \mathsf{bfq}(0, u + qt, qt, v) \\ \mathsf{bfq}(w, x, y, z) = \;& [w = 0 \wedge x = y \wedge z = 0] \\ & \vee [\exists n, qh : w = n \;\wedge\; x = qh \;\wedge\; \mathsf{bfq}(n, qh, y, z)] \\ & \vee [\exists n, l, r, e, qh, qt, es : \\ & \qquad w = n \;\wedge\; x = l + r + e + qh \;\wedge \\ & \qquad y = l + r + qt \;\wedge\; z = e + es \;\wedge \mathsf{bfq}(n, qh, qt, es)] \end{aligned}$$

It is easy to verify that $w = 0 \wedge x = y + z$ is a fixed point for bfq/4.

```
listof(X,Xs) :- Xs = [X].
listof(X,Xs) :- Xs = [X|Ys], listof(X,Ys).
```

Fig. 3. The listof/2 relation

To keep constraints in conjunctive form, it is common to interpret $\vee$ as a convex hull operation [3, 11]. Note that quantifier elimination can turn equality into inequality. For example, in $\mathsf{CLP}(\mathbb{N})$, $\exists z : x = y + z$ can be simplified to $y \leq x$. Inequalities may also be introduced by the convex hull operation.

Size relations analyses also provide groundness dependencies. Let us make this intuition precise, without delving in the details. Consider a substitution σ which is described by the conjunction of primitive $\mathsf{CLP}(\mathbb{N})$ constraints each of the form: $\sum_{j \in J} a_j x_j \leq b + \sum_{k \in K} b_k x_k$ where the coefficients a_j and b_k are positive. The groundness information contained in such a constraint can be expressed in Pos by: $\bigwedge_{k \in K} x_k \rightarrow \bigwedge_{j \in J} x_j$. For example, if σ satisfies the size relation $x \leq y + z$ and variables y and z become ground, then x must be ground as a result. Otherwise it would not be correct that *any instance* of σ would map x to a term smaller than $|y|_{\mathsf{vn}} + |z|_{\mathsf{vn}}$.

The groundness information we obtain through size analysis for bfq/4 in Figure 1 is identical to that obtained using Pos, namely $w \wedge (x \leftrightarrow y \wedge z)$. However, the size analysis of bf/2 is illuminating. It shows how Size may be more precise than Pos for groundness dependencies. The $\mathsf{CLP}(\mathbb{N})$ constraint is

$$\begin{aligned} \mathsf{bf}(u, v) &= \exists qt : \mathsf{bfq}(0, u + qt, qt, v) \\ &= \exists qt : 0 = 0 \wedge u + qt = qt + v \\ &= u = v \end{aligned}$$

That is, the groundness dependency $u \leftrightarrow v$ can be inferred. This is more precise than what was obtained with Pos. The Size solution $u = v$ was obtained because qt had the same coefficient on both sides of the equation $u + qt = qt + v$. With Pos we cannot distinguish this situation from $u + qt = qt + qt + v$, for example, and so the Pos result has to cover also the possible Size results $u \leq v$ and $u \geq v$.

Formalising the abstract interpretation between Size and Pos using the adjoint framework [9] for abstract interpretation is not obvious, since the translation described above does not give a best approximation. Consider for example the conjunction $(x + y + y = z) \ \wedge \ (x + y = z)$. The Pos information obtained from these constraints is $(x \wedge y) \leftrightarrow z$, while that obtained from the normalised system $x = z \wedge y = 0$ is the more precise $(x \leftrightarrow z) \wedge y$. So we need to define a normalisation procedure in order to use the adjoint framework. Alternatively we can use a more general framework, such as the relational framework [17].

However, Size is not the panacea for groundness analysis that it might appear to be. As an abstract domain it has infinite ascending chains, and hence we need to apply a widening operator in an analysis. Consider the program in Figure 3. An analysis using Pos finds the groundness dependency $x \leftrightarrow xs$. An analysis using Size constructs the equation:

$$\mathsf{listof}(x, xs) \quad = \quad xs = x \vee [\exists ys : xs = x + ys \wedge \mathsf{listof}(x, ys)]$$

Interpreting $\vee$ as the convex hull operation, Kleene iteration yields:

$$\begin{array}{lcl}
\mathsf{listof}^1(x, xs) & = & (xs = x) \\
\mathsf{listof}^2(x, xs) & = & (xs = x) \vee [\exists ys : xs = x + ys \wedge ys = x)] \ = \ (x \leq xs \leq 2x) \\
\mathsf{listof}^3(x, xs) & = & (x \leq xs \leq 3x) \\
\mathsf{listof}^4(x, xs) & = & (x \leq xs \leq 4x) \\
& \vdots &
\end{array}$$

To avoid traversing the infinite chain, it is customary to apply a widening operator which retains only the stable part of the constraint, and the result $x \leq xs$ is obtained as a fixed point. We have lost the groundness information $x \rightarrow xs$.

4 A More Precise Domain

Inspired by the Size and LSign domains, we introduce a new abstract domain, LPos, that solves the problems discussed in Sections 2 and 3. The idea is to represent linear constraints by their signs (abstract coefficients), but unlike LSign these abstract coefficients are introduced only by widening, and not at abstraction time. Basically, this enables us to preserve the Pos information which might be lost when applying the widening operation of the Size relations domain. For example, recall the analysis of the $\mathsf{listof}/2$ program in Size. The size relations analysis resulted in an infinite chain: $x \leq xs \leq x$, $x \leq xs \leq 2x$, $x \leq xs \leq 3x$, ... and so a widening was applied to get $x \leq xs$. With LPos the analysis gives $x \leq xs \leq \oplus x$, where $\oplus$ stands for some positive coefficient. This ensures the termination of the analysis and preserves the groundness dependencies otherwise lost by widening with Size.

4.1 The LPos domain

We define a primitive LPos constraint c to be of the form: $\sum_{i=1}^{n} a_i x_i \ op \ b$ where $a_i, b \in \mathbb{R} \cup \{\oplus, \ominus, \top\}$ and $op \in \{\leq, =\}$. Note that the coefficients in an LPos constraint can be concrete (rationals) or abstract (signs). The abstract coefficients represent sets of concrete coefficients under the obvious mapping: $\gamma_{\text{Sign}}(\top) = \mathbb{R}$, $\gamma_{\text{Sign}}(\oplus) = \{a \mid a \in \mathbb{R}, a > 0\}$, $\gamma_{\text{Sign}}(\ominus) = \{a \mid a \in \mathbb{R}, a < 0\}$ and $\gamma_{\text{Sign}}(a) = \{a\}$ otherwise. An LPos constraint is a conjunction of primitive LPos constraints.

The elements of LPos are (equivalence classes of) sets (for disjunction) of LPos constraints. The concretization function for LPos, γ_{LPos} maps the elements of LPos to sets of concrete constraints.

$$\begin{array}{r}
\gamma_{\mathsf{LPos}}(\sum_{i=1}^{n} a_i x_i \ op \ b) = \{\sum_{i=1}^{n} c_i x_i \ op \ d \mid c_i \in \gamma_{\text{Sign}}(a_i), d \in \gamma_{\text{Sign}}(b)\} \\
\gamma_{\mathsf{LPos}}(c_1 \wedge \cdots \wedge c_m) = \{d_1 \wedge \cdots \wedge d_m \mid d_i \in \gamma_{\mathsf{LPos}}(c_i), 1 \leq i \leq m\} \\
\gamma_{\mathsf{LPos}}(D) = \bigcup_{C \in D} \gamma_{\mathsf{LPos}}(C)
\end{array}$$

We consider concrete constraints modulo logical (constraint) equivalence. The concretization induces a partial order on LPos descriptions as follows:

$$D_1 \preceq_{\mathsf{LPos}} D_2 \quad \Leftrightarrow \quad \gamma_{\mathsf{LPos}}(D_1) \subseteq \gamma_{\mathsf{LPos}}(D_2)$$

The *join* and *meet* operations are defined as follows:

$$D_1 \sqcup_{\mathsf{LPos}} D_2 = D_1 \cup D_2 \quad \text{and} \quad D_1 \sqcap_{\mathsf{LPos}} D_2 = \{C_1 \wedge C_2 \mid C_1 \in D_1, C_2 \in D_2\}$$

Example 1. The LPos description $\{x \leq 0, 2x + \oplus y \leq 4 \wedge -x \leq 0\}$ represents the set of constraints $\{x \leq 0\} \cup \{2x + ay \leq 4 \wedge -x \leq 0 \mid a > 0\}$. Hence $2x + y \leq 4 \wedge -x \leq 0$ is represented by the LPos description, and, since it is logically equivalent, so is $2x + y \leq 4 \wedge -x \leq 0 \wedge y \leq 5 \wedge -x \leq 3$.

Abstract operations. The required operations on LPos to support groundness and uniqueness analyses are: *conjunction*, *disjunction* and *projection*. The first two are $\sqcap_{\mathsf{LPos}}$, and $\sqcup_{\mathsf{LPos}}$ respectively. Projection is the only complex operation in this domain. It is defined as an extension of Fourier elimination, enhanced to handle the abstract coefficients. Before defining the projection algorithm, we introduce arithmetic over $\mathbb{R} \cup \{\oplus, \ominus, \top\}$, and other operations needed for extracting the sign of the eliminated (by projection) variable's coefficient.

To eliminate variables we need to know their signs. Hence we define a mapping that maps coefficients to their signs:

Definition 2 (Sign abstraction, pos, neg).

$$\alpha_{\mathrm{Sign}}(a) = \begin{cases} 0 & \textit{if} \ \ a = 0 \\ \oplus & \textit{if} \ \ a > 0 \ \ \textit{or} \ \ a = \oplus \\ \ominus & \textit{if} \ \ a < 0 \ \ \textit{or} \ \ a = \ominus \\ \top & \textit{if} \ \ a = \top \end{cases}$$

$$pos(\sum_{i=1}^{n} a_i x_i \ op \ b) = \{x_i \mid \alpha_{\mathrm{Sign}}(a_i) = \oplus\}$$

$$neg(\sum_{i=1}^{n} a_i x_i \ op \ b) = \{x_i \mid \alpha_{\mathrm{Sign}}(a_i) = \ominus\}$$

Definition 3 (Arithmetic extension). *Let* $a_1, a_2 \in \mathbb{R} \cup \{\oplus, \ominus, \top\}$*,* $op \in \{\times, +\}$*. The results of the arithmetic operation* $a_1 \ op \ a_2$ *and of* $-a_1$ *(unary minus) are calculated using standard arithmetic if* a_1 *and* a_2 *are numbers, otherwise they are given as* $\alpha_{\mathrm{Sign}}(a_1) \ op \ \alpha_{\mathrm{Sign}}(a_2)$ *and* $-\alpha_{\mathrm{Sign}}(a_1)$ *according to these tables:*

$+$	0	$\oplus$	$\ominus$	$\top$
0	0	$\oplus$	$\ominus$	$\top$
$\oplus$	$\oplus$	$\oplus$	$\top$	$\top$
$\ominus$	$\ominus$	$\top$	$\ominus$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

$\times$	0	$\oplus$	$\ominus$	$\top$
0	0	0	0	0
$\oplus$	0	$\oplus$	$\ominus$	$\top$
$\ominus$	0	$\ominus$	$\oplus$	$\top$
$\top$	0	$\top$	$\top$	$\top$

$-$	
0	0
$\oplus$	$\ominus$
$\ominus$	$\oplus$
$\top$	$\top$

To obtain an accurate elimination procedure we need to know the sign of the coefficient of the variable we are eliminating. In case the coefficient is abstract and its sign is unknown (that is, $\top$), we *split* the constraint to obtain a disjunction of constraints in which that coefficient is known.

Definition 4 (Split). *The splitting function on a coefficient j is defined by*

$$\mathsf{split}(a) = \begin{cases} \{\oplus, \ominus, 0\} & \text{if } a = \top \\ \{a\} & \text{otherwise} \end{cases}$$
$$\mathsf{split}(\sum_{i=1}^{n} a_i x_i \; op \; b, j) = \{a x_j + \sum_{i=1, i \neq j}^{n} a_i x_i \; op \; b \mid a \in \mathsf{split}(a_j)\}$$
$$\mathsf{split}(c_1 \wedge \cdots \wedge c_m, j) = \{\bigwedge_{k=1}^{m} c'_k \mid c'_k \in \mathsf{split}(c_k, j)\}$$
$$\mathsf{split}(D, j) = \bigcup_{C \in D} \mathsf{split}(C, j)$$

Example 2. Given the LPos constraint C (with the constraint involving the coefficient $\top$ highlighted in a box):

$$(x_1 + 2x_2 \leq 2) \;\wedge\; (x_1 + \oplus x_2 \leq \ominus) \;\wedge\; \boxed{(\oplus x_1 + \top x_2 \leq 5)} \;\wedge\; (x_1 \leq 5)$$

the result of $\mathsf{split}(C, 2)$ is:

$$\left\{ \begin{array}{ll} C_{1,1}: & (x_1 + 2x_2 \leq 2) \;\wedge\; (x_1 + \oplus x_2 \leq \ominus) \;\wedge\; \boxed{(\oplus x_1 + \oplus x_2 \leq 5)} \;\wedge\; (x_1 \leq 5) \\ C_{1,2}: & (x_1 + 2x_2 \leq 2) \;\wedge\; (x_1 + \oplus x_2 \leq \ominus) \;\wedge\; \boxed{(\oplus x_1 + \ominus x_2 \leq 5)} \;\wedge\; (x_1 \leq 5) \\ C_{1,3}: & (x_1 + 2x_2 \leq 2) \;\wedge\; (x_1 + \oplus x_2 \leq \ominus) \;\wedge\; \boxed{(\oplus x_1 + 0 x_2 \leq 5)} \;\wedge\; (x_1 \leq 5) \end{array} \right\}$$

The constraints that were split are indicated in the boxes.

Clearly split does not change the interpretation of an abstract constraint:

Proposition 1. *For all j: $\gamma_{\mathsf{LPos}}(D) = \gamma_{\mathsf{LPos}}(\mathsf{split}(D, j))$.*

We can now define *projection*. $\mathsf{project}(D, V)$ projects an LPos description D, involving only inequalities, onto a set of variables V. Its definition is given in Figure 4. Note that we can always represent an LPos description using only inequalities by replacing an equality by two inequalities. Equalities can be handled directly by replacing $\mathsf{fourier}(C, j)$ by $\mathsf{fouriereq}(C, j)$.

Example 3. Consider the LPos description D:
$$\left\{ \begin{array}{ll} C_1: & x_1 + 2x_2 \leq 2 \;\wedge\; x_1 + \oplus x_2 \leq \ominus \;\wedge\; \oplus x_1 + \top x_2 \leq 5 \;\wedge\; x_1 \leq 5 \\ C_2: & x_1 \leq 0 \end{array} \right\}$$
The operation $\mathsf{project}(D, \{x_1\})$ proceeds by eliminating the variable x_2 as follows: First we apply $\mathsf{eliminate}(D, 2)$ which chooses C_1. Splitting C_1 gives the constraints described in Example 2. The call to $\mathsf{fourier}(C_{1,1}, 2)$ breaks up $C_{1,1}$ according to the sign of x_2, into

$C_\oplus = \left\{ \begin{array}{l} x_1 + 2x_2 \leq 2,\; x_1 + \oplus x_2 \leq \ominus, \\ \oplus x_1 + \oplus x_2 \leq 5 \end{array} \right\}$	$C_\ominus = \emptyset$	$C_0 = \{\, x_1 \leq 5 \,\}$

and yields $\{x_1 \leq 5\}$. $\mathsf{fourier}(C_{1,2}, 2)$ breaks up $C_{1,2}$ by the sign of x_2, into

$C_\oplus = \left\{ \begin{array}{l} x_1 + 2x_2 \leq 2 \\ x_1 + \oplus x_2 \leq \ominus \end{array} \right\}$	$C_\ominus = \{\, \oplus x_1 + \ominus x_2 \leq 5 \,\}$	$C_0 = \{\, x_1 \leq 5 \,\}$

project(D, V)
  $W := vars(D) - V$
  while $W \neq \emptyset$ do
    choose $x_j \in W$
    $W := W - \{x_j\}$
    $D :=$ eliminate(D, j)
  return D

eliminate(D, j)
  $D' := \emptyset$
  foreach $C \in D$
    foreach $C' \in$ split(C, j)
      $D' := D' \cup$ fourier(C', j)
  return D'

combine(c_1, c_2, j)
  let $\sum_{i=1}^{n} a_i^1 x_i \; op^1 \; b^1 \equiv c_1$
  let $\sum_{i=1}^{n} a_i^2 x_i \; op^2 \; b^2 \equiv c_2$
  if $\alpha_{\text{Sign}}(a_j^1 \times a_j^2) \neq \ominus$
    return $\emptyset$
  for $i := 1$ to n
    $a_i := a_i^1 \times (-a_j^2) + a_i^2 \times a_j^1$
  $a_j := 0$
  $b := b^1 \times (-a_j^2) + b^2 \times a_j^1$
  if $op^1 \equiv$ '$=$' and $op^2 \equiv$ '$=$'
    $op :=' ='$
  else
    $op :=' \leq'$
  return$\{\sum_{i=1}^{n} a_i x_i \; op \; b\}$

fourier(C, j)
  $C_0 := \{c \mid c \in C \text{ and } x_j \notin pos(c) \cup neg(c)\}$
  $C_\oplus := \{c \mid c \in C \text{ and } x_j \in pos(c)\}$
  $C_\ominus := \{c \mid c \in C \text{ and } x_j \in neg(c)\}$
  foreach $c_1 \in C_\oplus$
    foreach $c_2 \in C_\ominus$
      $C_0 := C_0 \cup$ combine(c_1, c_2, j)
  return C_0

negate(c)
  $c \equiv (\sum_{i=0}^{n} a_i x_i = b)$
  for $i := 1$ to n
    $a_i^1 := -a_i$
  $b^1 = -b$
  return $\sum_{i=0}^{n} a_i^1 x_i = b^1$

fouriereq(C, j)
  $C_0 := \{c \mid c \in C \text{ and } x_j \notin pos(c) \cup neg(c)\}$
  $C_\oplus := \{c \mid c \in C \text{ and } x_j \in pos(c)\}$
  $C_\ominus := \{c \mid c \in C \text{ and } x_j \in neg(c)\}$
  foreach $c_1 \in (C_\oplus \cup C_\ominus)$
    let $\sum_{i=1}^{n} a_i^1 x_i \; op \; b^1 \equiv c_1$
    $c_1^+ := c_1; c_1^- := c_1;$
    if $op \equiv' ='$
      if $\alpha_{\text{Sign}}(a_j^1) = \oplus$ $\quad c_1^- :=$ negate(c_1)
      else $\quad c_1^+ :=$ negate(c_1)
    foreach $c_2 \in (C_\oplus \cup C_\ominus) - \{c_1\}$
      $c_2 \equiv \sum_{i=1}^{n} a_i^2 x_i \; op^2 \; b^2$
      if $\alpha_{\text{Sign}}(a_j^2) = \oplus$
        $C_0 = C_0 \cup$ combine(c_2, c_1^-, j)
      else $C_0 = C_0 \cup$ combine(c_1^+, c_2, j)
  return C_0

Fig. 4. Projection algorithm for LPos

and returns $\{x_1 \leq 5 \wedge \oplus x_1 \leq \oplus \wedge \oplus x_1 \leq \top\}$. Now, the call to fourier$(C_{1,3}, 2)$ breaks up $C_{1,3}$ by the sign of x_2, into

$C_\oplus = \left\{ \begin{array}{l} x_1 + 2x_2 \leq 2, \\ x_1 + \oplus x_2 \leq \ominus \end{array} \right\}$	$C_\ominus = \emptyset$	$C_0 = \{\, x_1 \leq 5, \oplus x_1 + 0x_2 \leq 5 \,\}$

and it yields $\{x_1 \leq 5 \wedge \oplus x_1 \leq 5\}$. The returned values from fourier, together with $x_1 \leq 0$ (the result for C_2) make up the result of eliminate:

$$\left\{\begin{array}{ll} x_1 \leq 5, & x_1 \leq 5 \wedge \oplus x_1 \leq \oplus \wedge \oplus x_1 \leq \top \\ x_1 \leq 5 \wedge \oplus x_1 \leq 5, & x_1 \leq 0 \end{array}\right\}$$

Widening. The LPos domain contains infinite ascending chains, so we need to use widening. The idea behind the widening is to join abstract primitive constraints in such as way that no groundness or uniqueness dependency is lost.

Definition 5 (Sign equivalence). *Given two abstract primitive* LPos *constraints:*

$$c_1 \equiv \sum_{i=0}^{n} a_i^1 x_i \ op_1 \ b_1 \qquad c_2 \equiv \sum_{i=0}^{n} a_i^2 x_i \ op_2 \ b_2$$

we say that c_1 *and* c_2 *are sign equivalent, denoted* $c_1 \sim c_2$ *if and only if:* $op_1 \equiv op_2$*,* $\alpha_{\mathrm{Sign}}(a_i^1) \equiv \alpha_{\mathrm{Sign}}(a_i^2)$ $(0 \leq i \leq n)$*, and* $\alpha_{\mathrm{Sign}}(b_1) \equiv \alpha_{\mathrm{Sign}}(b_2)$*. Two* LPos *constraints (conjunctions)* C_1 *and* C_2 *are sign equivalent, if for each* $c_1 \in C_1$ *there is a* $c_2 \in C_2$ *such that* $c_1 \sim c_2$*, and vice versa.*

Example 4. Consider the following two abstract LPos constraints:

$$\begin{array}{lcl} C_1 & \equiv & \overbrace{x + \oplus y \leq 10}^{c_1} \ \wedge \ \overbrace{x - y \leq 5}^{c_2} \\ C_2 & \equiv & \overbrace{x + 4y \leq 4}^{c_3} \ \wedge \ \overbrace{x + \ominus y \leq 1}^{c_4} \end{array}$$

c_1, c_3 and c_2, c_4 are sign equivalent, so C_1 and C_2 are sign equivalent. If we add $c_5 \equiv x \leq \oplus$ to C_1, then C_1 and C_2 are no longer sign equivalent, because c_5 is not sign equivalent with c_3 or c_4.

Proposition 2. *Given a finite set of variables* $\mathcal{V}$*, there are only finitely many non-sign equivalent abstract* LPos *constraints.*

The "widening" that we apply is based on the notion of sign equivalence. This operation is applied when two constraints are sign equivalent, so we do not lose the sign information. This is important for groundness and uniqueness analysis.

Definition 6 (Widening coefficients). *Let* $a_1, a_2 \in \mathbb{R} \cup \{\oplus, \ominus, \top\}$*. Define* $a_1 \sqcup a_2 \stackrel{def}{=} a_1$ *if* $a_1 = a_2$*, otherwise* $a_1 \sqcup a_2 \stackrel{def}{=} \alpha_{\mathrm{Sign}}(a_1)$ *if* $\alpha_{\mathrm{Sign}}(a_1) = \alpha_{\mathrm{Sign}}(a_2)$*, otherwise* $a_1 \sqcup a_2 = \top$*. Extend this to sign equivalent constraints:*

$$\left[\sum_{i=1}^{n} a_i^1 x_i \ op \ b_1\right] \ \sqcup \ \left[\sum_{i=1}^{n} a_i^2 x_i \ op \ b_2\right] = \sum_{i=1}^{n} (a_i^1 \sqcup a_i^2) x_i \ op \ (b_1 \sqcup b_2)$$
$$(c_1^1 \wedge \cdots c_n^1) \sqcup (c_1^2 \wedge \cdots c_m^2) = \bigwedge \{c_i^1 \sqcup c_j \mid c_i^1 \sim c_j^2\}$$

Finally we obtain a widening operation, by widening sign equivalent constraints in the descriptions.

$$\begin{array}{rl} \mathsf{widen}(D_1, D_2) = & (D_1 \setminus \{C_1 \mid C_1 \in D_1, \exists C_2 \in D_2, C_1 \sim C_2\}) \\ & \cup \ (D_2 \setminus \{C_2 \mid C_2 \in D_2, \exists C_1 \in D_1, C_1 \sim C_2\}) \\ & \cup \ \{C_1 \sqcup C_2 \mid C_1 \in D_1, C_2 \in D_2, C_1 \sim C_2\} \end{array}$$

Example 5. Let LPos constraints C_1 and C_2 be as in Example 4. Then

$$\mathsf{widen}(\{C_1\}, \{C_2\}) = \{x + \oplus y \leq \oplus \wedge x + \ominus y \leq \oplus\}.$$

5 LPos instead of Pos

In this section we describe the use of LPos for groundness and uniqueness analysis, and show that it is uniformly more accurate than the corresponding Pos analyses. Before going into details we define a splitting function on all variables, which is based on the split given in Definition 4. We use this to map the LPos elements to their Pos description, in both groundness and uniqueness analysis. The split function effectively replaces all $\top$ coefficients by the three choices they represent $\{\oplus, \ominus, 0\}$.

Definition 7 (Splitting on all variables). *Given an* LPos *conjunction C, we define the splitting of C on all its variables as follows:*

$$\begin{aligned}
\mathsf{splitall}(c_1 \wedge \cdots \wedge c_m) &= \{c'_1 \wedge \cdots \wedge c'_m \mid c'_i \in \mathsf{splitall}(c_i)\}\\
\mathsf{splitall}(\textstyle\sum_{i=1}^{n} a_i x_i \ op\ b) &= \mathsf{splitall}^*(\textstyle\sum_{i=1}^{n} a_i x_i \ op\ b, n)\\
\mathsf{splitall}^*(c,n) &= \begin{cases} \{c\} & \textit{if } \ n = 0\\ \bigcup\limits_{c' \in \mathsf{split}(c,n)} \mathsf{splitall}^*(c', n-1) & \textit{otherwise} \end{cases}
\end{aligned}$$

Example 6. Consider the LPos constraint $\oplus x + \top y \leq \oplus \wedge \top x + \ominus y = \ominus$. Splitting on all variables, that is, on x and y, gives a set of nine abstract constraints:

$$\mathsf{splitall}(C) =$$
$$\left\{\begin{array}{ll}
\oplus x + 0y \leq \oplus \ \wedge 0x + \ominus y = \ominus, & \oplus x + \oplus y \leq \oplus \wedge 0x + \ominus y = \ominus\\
\oplus x + \ominus y \leq \oplus \wedge 0x + \ominus y = \ominus, & \oplus x + 0y \leq \oplus \ \wedge \oplus x + \ominus y = \ominus\\
\oplus x + \oplus y \leq \oplus \wedge \oplus x + \ominus y = \ominus, & \oplus x + \ominus y \leq \oplus \wedge \oplus x + \ominus y = \ominus\\
\oplus x + 0y \leq \oplus \ \wedge \ominus x + \ominus y = \ominus, & \oplus x + \oplus y \leq \oplus \wedge \ominus x + \ominus y = \ominus\\
\oplus x + \ominus y \leq \oplus \wedge \ominus x + \ominus y = \ominus &
\end{array}\right\}$$

By Proposition 1 we can eliminate $\top$ coefficients using splitall. From here on we therefore assume that no LPos description involves $\top$.

Groundness analysis of logic programs: Given a logic program P, the meaning of P over the LPos domain is obtained in two stages: the first is abstracting the Herbrand terms to abstract LPos constraints using the var-norm function, and the second is computing the meaning of the program. For simplicity, we will use a Kleene iteration combined with *widening*, in order to compute an approximation of the least fixed point of the equation generated by the abstraction function.

The abstract LPos version P' of a normalised logic program P is obtained as follows: For each clause $C \equiv p(\overline{x}) \leftarrow b_1, \ldots, b_n$ we generate an equation:

$$p(\overline{x}) = \overline{\exists}\overline{x} : b'_1 \wedge \ldots \wedge b'_n \wedge \bigwedge_{v \in vars(C)} (v \geq 0)$$

where each b'_i is a translation of b_i according to this rule: $b'_i \equiv \mathsf{var\text{-}norm}(s) \leq \mathsf{var\text{-}norm}(t) \wedge \mathsf{var\text{-}norm}(t) \leq \mathsf{var\text{-}norm}(s)$ if $b_i \equiv (s = t)$, otherwise $b'_i = b_i$. The operations $\wedge$ and $\overline{\exists}$ are $\sqcap_{\mathsf{LPos}}$ and $\mathsf{project}$ respectively. The *join* operation will be denoted by $\vee$. The Kleene iteration is performed in the standard way, but after each step (or finitely many steps) we apply a widening.

Similar to Size, LPos objects describe groundness and can be translated to Pos descriptions according to the description function defined below.

Definition 8 (Description function $\alpha_{\mathcal{G}}$). *We can extract groundness information from an* LPos *object as follows:*

$$\alpha_{\mathcal{G}}(c) = \begin{cases} \bigvee\limits_{c' \in \mathsf{splitall}(c)} (\bigwedge neg(c') \;\rightarrow\; \bigwedge pos(c')) & \text{if } c \text{ is an inequality} \\ \bigvee\limits_{c' \in \mathsf{splitall}(c)} (\bigwedge neg(c') \;\leftrightarrow\; \bigwedge pos(c')) & \text{if } c \text{ is an equation} \end{cases}$$

$$\alpha_{\mathcal{G}}(C) = \bigwedge_{i=1}^{n} \alpha_{\mathcal{G}}(c_i), \quad \text{where } C \equiv c_1 \wedge \cdots \wedge c_n$$

$$\alpha_{\mathcal{G}}(D) = \bigvee_{C \in D} \alpha_{\mathcal{G}}(C)$$

Example 7. Let us (re)analyse the listof program from Figure 3, this time using LPos, and show how LPos gives better information than Size. First we use the var-norm function to abstract the program to the following recursive equations:

$$\begin{aligned} \mathsf{listof}(x, xs) = \;& -x \leq 0 \wedge -xs \leq 0 \wedge xs = x \\ & \vee [\exists ys : -x \leq 0 \wedge -xs \leq 0 \wedge -ys \leq 0 \wedge xs = x + ys \wedge \mathsf{listof}(x, ys)] \end{aligned}$$

and two steps of Kleene iteration yield:

$$\begin{aligned} \mathsf{listof}^1(x, xs) &= [-x \leq 0 \wedge -xs \leq 0 \wedge xs = x] \\ \mathsf{listof}^2(x, xs) &= [-x \leq 0 \wedge -xs \leq 0 \wedge xs = x] \\ &\quad \vee [\exists ys : -x \leq 0 \wedge -xs \leq 0 \wedge -ys \leq 0 \wedge xs = x + ys \wedge ys = x)] \\ &= [-x \leq 0 \wedge -xs \leq 0 \wedge xs = x] \vee [-x \leq 0 \wedge -xs \leq 0 \wedge xs = x + x] \end{aligned}$$

Applying a widening on $\mathsf{listof}^1(x, xs)$ and $\mathsf{listof}^2(x, xs)$ gives

$$C = -x \leq 0 \wedge -xs \leq 0 \wedge xs = \oplus x$$

which is a fixed point, and its Pos description is $\alpha_{\mathcal{G}}(C) = xs \leftrightarrow x$.

Lemma 1. *Let D_1, D_2 be elements of* LPos *and $\mathcal{V}$ a set of variables. Then:* **(a)** $\alpha_{\mathcal{G}}(D_1 \sqcup_{\mathsf{LPos}} D_2) \Leftrightarrow \alpha_{\mathcal{G}}(D_1) \vee \alpha_{\mathcal{G}}(D_2)$ *and* $\alpha_{\mathcal{G}}(D_1 \sqcap_{\mathsf{LPos}} D_2) \Leftrightarrow \alpha_{\mathcal{G}}(D_1) \wedge \alpha_{\mathcal{G}}(D_2)$*;* **(b)** $\alpha_{\mathcal{G}}(\mathsf{project}(D_1, \mathcal{V})) \Rightarrow \overline{\exists}\mathcal{V} : \alpha_{\mathcal{G}}(D_1)$*; and* **(c)** $\alpha_{\mathcal{G}}(D_1 \sqcup_{\mathsf{LPos}} D_2) = \alpha_{\mathcal{G}}(\mathsf{widen}(D_1, D_2))$.

Theorem 1. **(a)** *$\alpha_{\mathcal{G}}$ correctly (and accurately) extracts groundness information from* LPos *descriptions; and* **(b)** LPos *is uniformly at least as accurate as* Pos *for groundness analysis of logic programs.*

Uniqueness analysis: With LPos, the analysis of a constraint program has two stages, just like groundness analysis has. Only the abstraction function differs. Given a normalised linear arithmetic constraint program P, an abstract LPos version P' is obtained as follows: For each clause $p(\overline{x}) \leftarrow b_1, \ldots, b_n$ generate the equation: $p(\overline{x}) = \overline{\exists}\overline{x} : b'_1 \wedge \ldots \wedge b'_n$ where b'_i is a translation of b_i: if b_i is an equation or a call, $b'_i = b_i$, otherwise $b'_i = true$. The operations $\wedge$, $\vee$ and $\overline{\exists}$ are the same as in groundness analysis, And Kleene iteration is performed similarly. It is clear that the LPos object obtained in this analysis, is an approximation of the concrete object of P, so a uniqueness analysis is obtained by using the following description function. below.

Definition 9 (Description function $\alpha_{\mathcal{U}}$). *We can extract uniqueness information from an* LPos *objects as follows:*

$$\begin{aligned}
\alpha_{\mathcal{U}}(c) &= \begin{cases} true & \text{if C is an inequality} \\ \bigvee_{c \in \mathsf{splitall}(C)} \nabla(vars(c)) & \text{if C is an equation} \end{cases} \\
\alpha_{\mathcal{U}}(C) &= \bigwedge_{i=1}^{n} \alpha_{\mathcal{U}}(c_i), \quad \text{where } C \equiv c_1 \wedge \cdots \wedge c_n \\
\alpha_{\mathcal{U}}(D) &= \bigvee_{C \in D} \alpha_{\mathcal{U}}(C)
\end{aligned}$$

Example 8. Using LPos, the abstract version of Figure 2 is

$$\begin{aligned}
\mathsf{mg}(p,t,r,b) = \quad & [(t=1) \wedge (b = cp - r)] \\
\vee\, & [\exists t_1, p_1 : (t_1 = t-1) \wedge (p_1 = cp - r) \wedge \ \mathsf{mg}(p_1, t_1, r, b)]
\end{aligned}$$

Kleene iteration proceeds as follows:

$$\begin{aligned}
\mathsf{mg}^1(p,t,r,b) = \quad & [(t=1) \wedge (b = cp - r)] \\
\mathsf{mg}^2(p,t,r,b) = \quad & [(t=1) \wedge (b = cp - r)] \\
\vee\, & [\exists t_1, p_1 : (t_1 = t-1) \wedge (p_1 = cp - r) \wedge \ (t_1 = 1) \wedge (b = cp_1 - r)] \\
= \quad & [(t=1) \wedge (b = cp - r)] \\
\vee\, & [(t=2) \wedge (b + (1+c)r - c^2 p) = 0]
\end{aligned}$$

and a widening on $\mathsf{mg}^1(p,t,r,b)$ and $\mathsf{mg}^2(p,t,r,b)$ yields:

$$\mathsf{mg}^3(p,t,r,b) \ = \ (t = \oplus) \wedge (b + \oplus r + \ominus p = 0)$$

which is a fixed point. Its description in Pos is

$$t \wedge ((p \wedge r) \rightarrow b) \wedge ((b \wedge r) \rightarrow p) \wedge ((b \wedge p) \rightarrow r)$$

which is more precise than the one we obtained using Pos, namely $t \wedge ((p \wedge r) \rightarrow b)$.

Lemma 2. *Let D_1 and D_2 be elements of* LPos *and $\mathcal{V}$ a set of variables. Then:* **(a)** $\alpha_{\mathcal{U}}(D_1 \sqcup_{\mathsf{LPos}} D_2) \Rightarrow \alpha_{\mathcal{U}}(D_1) \vee \alpha_{\mathcal{U}}(D_2)$ *and* $\alpha_{\mathcal{U}}(D_1 \sqcap_{\mathsf{LPos}} D_2) \Rightarrow \alpha_{\mathcal{U}}(D_1) \wedge \alpha_{\mathcal{U}}(D_2)$*;* **(b)** $\alpha_{\mathcal{U}}(\mathsf{project}(D_1, \mathcal{V})) \Rightarrow \overline{\exists}\mathcal{V} : \alpha_{\mathcal{U}}(D_1)$*; and* **(c)** $\alpha_{\mathcal{U}}(D_1 \sqcup_{\mathsf{LPos}} D_2) = \alpha_{\mathcal{U}}(\mathsf{widen}(D_1, D_2))$.

Theorem 2. **(a)** *$\alpha_{\mathcal{U}}$ correctly (and accurately) extracts uniqueness information from* LPos *descriptions; and* **(b)** LPos *is uniformly at least as accurate as* Pos *for uniqueness analysis of* $\mathsf{CLP}(\mathbb{R}_{\mathsf{Lin}})$ *programs.*

6 Conclusion

We have shown how the well-known Pos domain occasionally is less than adequate for groundness and uniqueness analysis. For example, a Pos-based analysis of the celebrated "mortgage" program exhibits a significant loss of precision.

A partial solution can be obtained by using instead a domain Size of term size constraints. However, we have shown that this solution is not entirely satisfactory. In fact even the reduced product [10] of Pos and Size has shortcomings for uniqueness analysis. For this reason, inspired by the LSign domain, we proposed LPos as an improved Pos for groundness and uniqueness analysis.

A different attempt to improve the precision of Pos-based analysis was made by Filé and Ranzato [13] who considered the use of the powerset of Pos, $\mathcal{P}(\mathsf{Pos})$. However, while this increases the expressiveness of the abstract domain, it can be argued that groundness analysis cannot capitalise on the improved precision, since the Pos content of the result of a $\mathcal{P}(\mathsf{Pos})$ analysis is exactly that of a Pos analysis [14]. (This is interesting because it differs from the situation where a smaller domain is chosen as starting point, for example, the result of a Def analysis [1] may be a less precise than the Def content of a $\mathcal{P}(\mathsf{Def})$ analysis).

Marriott and Stuckey originally introduced the LSign domain [20] to derive information about the satisfiability of linear arithmetic constraints. LSign could be used for groundness and uniqueness analysis also, but it is less precise than what we can obtain in LPos. LPos extends the original LSign in the same direction as the LInt interval improvement of LSign [21], by using both concrete coefficients (size 0 intervals) and abstract coefficients (semi-infinite intervals) and sets of abstract constraints. Unlike LSign and LInt, LPos does not use abstract Gauss-Jordan variable elimination. Interestingly, using this elimination algorithm, the accuracy theorems *fail* to hold. The Fourier elimination creates redundant constraints, and these are anathema to the LSign goal of detecting satisfiability, but they do not reduce the accuracy of the resulting LPos descriptions.

References

1. T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean functions for dependency analysis. *Science of Computer Programming*, 31(1):3–45, 1998.
2. N. Baker and H. Søndergaard. Definiteness analysis for CLP($\mathcal{R}$). In G. Gupta *et al.*, editor, *Proc. 16th Australian Computer Science Conf.*, pages 321–332, 1993.
3. F. Benoy and A. King. Inferring argument size relationships with CLP($\mathcal{R}$). In J. Gallagher, editor, *Logic Program Synthesis and Transformation*, volume 1207 of *Lecture Notes in Computer Science*, pages 204–223. Springer, 1997.
4. M. Codish and B. Demoen. Deriving polymorphic type dependencies for logic programs using multiple incarnations of Prop. In B. Le Charlier, editor, *Static Analysis: Proc. First Int. Symp.*, volume 864 of *Lecture Notes in Computer Science*, pages 281–296. Springer, 1994.
5. M. Codish and C. Taboch. A semantic basis for the termination analysis of logic programs. *J. Logic Programming*, 41(1):103–123, 1999.

6. P. Codognet and G. Filé. Computations, abstractions and constraints in logic programs. In *Proc. 4th IEEE Int. Conf. Computer Languages*, pages 155–164. IEEE Comp. Soc. Press, 1992.
7. A. Cortesi, G. Filé, and W. Winsborough. *Prop* revisited: Propositional formula as abstract domain for groundness analysis. In *Proc. 6th Annual IEEE Symp. Logic in Computer Science*, pages 322–327. IEEE Computer Society Press, 1991.
8. A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Evaluation of the domain Prop. *Journal of Logic Programming*, 23(3):237–278, 1995.
9. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th Ann. ACM Symp. Principles of Programming Languages*, pages 238–252. ACM Press, 1977.
10. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th Ann. ACM Symp. Principles of Programming Languages*, pages 269–282. ACM Press, 1979.
11. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Programming Language Implementation and Logic Programming*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295. Springer, 1992.
12. C. Fecht. *Abstrakte Interpretation logischer Programme: Theorie, Implementierung, Generierung*. PhD thesis, Universität des Saarlandes, Germany, 1997.
13. G. Filé and F. Ranzato. Improving abstract interpretations by systematic lifting to the powerset. In M. Bruynooghe, editor, *Logic Programming: Proc. 1994 Int. Symp.*, pages 655–669. MIT Press, 1994.
14. G. Filé and F. Ranzato. The powerset operator on abstract interpretations. *Theoretical Computer Science*, 222:77–111, 1999.
15. R. Giacobazzi, S. Debray, and G. Levi. Generalized semantics and abstract interpretation for constraint logic programs. *J. Logic Programming*, 25(3):191–247, 1995.
16. N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In L. Naish, editor, *Logic Programming: Proc. 14th Int. Conf. Logic Programming*, pages 63–77. MIT Press, 1997.
17. K. Marriott. Frameworks for abstract interpretation. *Acta Informatica*, 30(2):103–129, 1993.
18. K. Marriott and H. Søndergaard. Notes for a tutorial on abstract interpretation of logic programs. 1989 North American Conf. Logic Programming.
19. K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM Letters on Programming Languages and Systems*, 2(1–4):181–196, 1993.
20. K. Marriott and P. J. Stuckey. Approximating interaction between linear arithmetic constraints. In M. Bruynooghe, editor, *Logic Programming: Proc. 1994 Int. Symp.*, pages 571–585. MIT Press, 1994.
21. V. Ramachandran, P. Van Hentenryck, and A. Cortesi. Abstract domains for reordering CLP($\Re_{Lin}$) programs. *J. Logic Programming*, 42(3):217–256, 2000.
22. Peter Schachte. *Precise and Efficient Static Analysis of Logic Programs.* PhD thesis, The University of Melbourne, Australia, 1999.
23. A. Van Gelder. Deriving constraints among argument sizes in logic programs. In *Proc. 9th ACM Conf. Principles of Database Systems*, pages 47–60. ACM Press, 1990.