Parameterizing a Groundness Analysis of Logic Programs

Lunjin Lu

Department of Computer Science and Engineering
Oakland University
Rochester, MI 48309, USA
EMail: l2lu@oakland.edu
Phone: +248-370-2225
FAX: +248-370-4625

Abstract. We present a parametric groundness analysis whose input and output are parameterized by a set of groundness parameters. The result of the analysis can be instantiated for different uses of the program. It can also be used to derive sufficient conditions for safely removing groundness checks for built-in calls in the program. The parametric groundness analysis is obtained by generalizing a non-parametric groundness analysis that uses the abstract domain Con. It is shown to be as precise as the non-parametric groundness analysis for any possible values for the groundness parameters. Experimental results of a prototype implementation of the parametric groundness analysis are given.
Keywords: Abstract Interpretation, Groundness Analysis, Logic programs

1 Introduction

In logic programming [30, 1], a real world problem is modeled as a set of axioms and a general execution mechanism is used to solve the problem. While this allows a problem to be solved in a natural and declarative manner, the general execution mechanism incurs a performance penalty for most programs. This motivated much research into semantic based analysis of logic programs [20]. Groundness analysis is one of the most important analyses for logic programs. It provides answers to questions such as whether, at a program point, a variable is definitely bound to a ground term - a term that contains no variables. This is useful not only to an optimizing compiler but also to other program manipulation tools. There have been many methods proposed for groundness analysis [36, 43, 22, 6, 14, 46, 17, 34, 17, 2, 3, 29, 25, 9, 8, 44].

This paper presents a new groundness analysis whose input and output are parameterized by a number of groundness parameters, hence called parametric groundness analysis. These parameters represent groundness information that is not available before analysis but can be provided after analysis. Providing such information *instantiates* the result of analysis. Instantiability implies reusability. A program module such as a library program can be analyzed once and the result be instantiated for different uses of the program module. This improves the efficiency of analysis. Instantiability also makes the new groundness analysis amenable to program modifications since modules that are not changed need not be re-analyzed. Groundness parameters in the input and the output of the new groundness analysis makes it easier to derive a sufficient condition under which groundness checks for built-in calls in the program can be safely removed.

The parametric groundness analysis is obtained by generalizing a groundness analysis based on the abstract domain Con [43]. Con is the least precise abstract domain for groundness analysis. The parametric groundness analysis is thus less precise than a groundness analysis that uses a more precise abstract domain namely Pos [34], Def [22] or EPos [24]. However, a Con-based groundness analysis is much more efficient than groundness analyzers based on more precise abstract domains. By generalizing a Con-based groundness analysis, we obtain a parametric groundness analysis that is more efficient and scalable.

The parametric groundness analysis is performed by abstract interpretation [18, 19]. Abstract interpretation is a methodology for static program analysis whereby a program analysis is viewed as the execution of the program over a non-standard data domain. A number of frameworks have been brought about for abstract interpretation of logic programs [37, 5, 6, 27, 28, 35, 39, 33]. An abstract interpretation framework is an analysis engine that takes care of algorithmic issues that are common to a class of analyses, allowing the designer of an analysis to focus on issues that are specific to the analysis. This greatly simplifies the design and the presentation of a new analysis. The parametric groundness analysis will be presented in the abstract interpretation framework in [31]. The adaptation to other frameworks [37, 5, 6, 27, 28, 35, 39, 33] can be easily made.

The remainder of the paper is organized as follows. Section 2 gives motivation for the parametric groundness analysis through an example. Section 3 gives basic notations and briefly describes the abstract interpretation framework in [31]. Section 4 reformulates a non-parametric groundness analysis that is generalized in section 5 to obtain the new groundness analysis. Section 6 provides performance results of a prototype implementation. In section 7, we compare our work with related work. Section 8 concludes the paper. Proofs are omitted due to space limit.

2 Motivation

In groundness analysis, we are interested in knowing which variables will be definitely instantiated to ground terms and which variables are not when the execution of the program reaches a program point. We use g and u to represent these two groundness modes of a variable. Let $\mathsf{MO} \stackrel{def}{=} \{\mathsf{g}, \mathsf{u}\}$ and $\trianglelefteq$ be defined as $\mathsf{g}\trianglelefteq\mathsf{g}$, $\mathsf{g}\trianglelefteq\mathsf{u}$ and $\mathsf{u}\trianglelefteq\mathsf{u}$. $\langle\mathsf{MO}, \trianglelefteq\rangle$ is a complete lattice with infimum g and supremum u. Let $\triangledown$ and $\triangle$ be the least upper bound and the greatest lower bound operators on $\langle\mathsf{MO}, \trianglelefteq\rangle$ respectively.

Example 1. Consider the program and the initial goal in Figure 1. Let ⓐ : $Q \Longrightarrow$ ⓑ : R denote that if Q holds at the program point ⓐ then R holds whenever the

$\leftarrow$ ⓐ $treesort(Li, Lo).$ ⓑ

$treesort(Li, Lo) \leftarrow list\_to\_tree(Li, T), tree\_to\_list(T, Lo).$

$insert(I, void, tr(I, void, void)).$
$insert(I, tr(E, L, R), tr(E, Ln, R)) \leftarrow$ ⓒ $I < E, insert(I, L, Ln).$
$insert(I, tr(E, L, R), tr(E, L, Rn)) \leftarrow$ ⓓ $I >= E, insert(I, R, Rn).$

$insert\_list([H|L], T, Tn) \leftarrow insert(H, T, Tm), insert\_list(L, Tm, Tn).$
$insert\_list([], T, T).$

$list\_to\_tree(L, T) \leftarrow insert\_list(L, void, T).$

$tree\_to\_list(T, L) \leftarrow tree\_to\_list\_aux(T, [], L).$

$tree\_to\_list\_aux(void, L, L).$
$tree\_to\_list\_aux(tr(I, L, R), O, N) \leftarrow$
$\quad tree\_to\_list\_aux(R, O, L1), tree\_to\_list\_aux(L, [I|L1], N).$

Fig. 1. The treesort program from [21]. Circled letters are not part of the program but locate program points.

execution reaches the program point ⓑ. Let $X \mapsto \mathsf{m}$ denote that the groundness mode of X is m. A groundness analysis infers the following statements.

$$\begin{aligned}
ⓐ : (Li \mapsto \mathsf{g}) \wedge (Lo \mapsto \mathsf{g}) &\Longrightarrow ⓒ : (I \mapsto \mathsf{g}) \wedge (E \mapsto \mathsf{g})\\
ⓐ : (Li \mapsto \mathsf{g}) \wedge (Lo \mapsto \mathsf{u}) &\Longrightarrow ⓒ : (I \mapsto \mathsf{g}) \wedge (E \mapsto \mathsf{g})\\
ⓐ : (Li \mapsto \mathsf{u}) \wedge (Lo \mapsto \mathsf{g}) &\Longrightarrow ⓒ : (I \mapsto \mathsf{u}) \wedge (E \mapsto \mathsf{u})\\
ⓐ : (Li \mapsto \mathsf{u}) \wedge (Lo \mapsto \mathsf{u}) &\Longrightarrow ⓒ : (I \mapsto \mathsf{u}) \wedge (E \mapsto \mathsf{u})
\end{aligned}$$

These statements must be inferred independently from each other. The groundness of I and E at the point ⓒ depends on the groundness of Li and Lo at the point ⓐ in such a way that I and E are ground at point ⓒ iff Li is ground at point ⓐ. Thus, it will be desirable to have a groundness analysis which infers the following statement

$$ⓐ : (Li \mapsto \alpha) \wedge (Lo \mapsto \beta) \Longrightarrow ⓒ : (I \mapsto \alpha) \wedge (E \mapsto \alpha) \qquad (1)$$

where α and β are groundness parameters ranging over MO. ∎

Such an analysis is parametric in the sense that its input ⓐ : $(Li \mapsto \alpha) \wedge (Lo \mapsto \beta)$ and its output ⓒ: $(I \mapsto \alpha) \wedge (E \mapsto \alpha)$ are parameterized.

Statement (1) can be instantiated as follows. When the parameters α and β are assigned groundness modes from MO, the groundness of Li and Lo at the point ⓐ and the groundness of I and E at the point ⓒ are obtained by instantiation. The first statement inferred by the non-parametric groundness analysis is obtained from (1) by assigning g to both α and β. Instantiations can be made for four different assignments of groundness modes to α and β.

Statement (1) can also be used to infer a sufficient condition on α and β under which the run-time check on the groundness of I and E in the built-in call $I < E$ at the point ⓒ can be safely removed. Specifically, if α is assigned g then the run-time check can be safely removed. With a non-parametric groundness analysis, one needs to analyze the program for four times to infer the sufficient condition.

3 Preliminaries

Lattice Theory A poset is a tuple $\langle A, \sqsubseteq \rangle$ where A is a set and $\sqsubseteq$ is a reflexive, anti-symmetric and transitive relation on A. Let $B \subseteq A$ and $u \in A$. u is an upper bound of B if $b \sqsubseteq u$ for each $b \in B$. u is a least upper bound of B if $u \sqsubseteq u'$ for any upper bound u' of B. The least upper bound of B, if exists, is unique and denoted $\sqcup B$. Lower bounds and the greatest lower bound are defined dually. $\sqcap B$ denotes the greatest lower bound of B.

A complete lattice is a poset $\langle A, \sqsubseteq \rangle$ such that $\sqcup B$ and $\sqcap B$ exist for any $B \subseteq A$. A complete lattice is denoted $\langle A, \sqsubseteq, \bot, \top, \sqcap, \sqcup \rangle$ where $\bot \stackrel{def}{=} \sqcup\emptyset$ and $\top \stackrel{def}{=} \sqcap\emptyset$. Let $\langle A, \sqsubseteq, \bot, \top, \sqcap, \sqcup \rangle$ be a complete lattice and $B \subseteq A$. B is a Moore family if $\top \in B$ and $(x_1 \sqcap x_2) \in B$ for any $x_1 \in B$ and $x_2 \in B$.

Let $\langle A, \sqsubseteq_A \rangle$ and $\langle B, \sqsubseteq_B \rangle$ be two posets. A function $f : A \mapsto B$ is monotonic if $f(a_1) \sqsubseteq_B f(a_2)$ for any $a_1 \in A$ and $a_2 \in A$ such that $a_1 \sqsubseteq_A a_2$. Let $X \subseteq A$. We define $f(X) \stackrel{def}{=} \{f(x) \mid x \in X\}$. We sometimes use Church's lambda notation for functions, so that a function f will be denoted $\lambda x.f(x)$.

Logic programming Let Σ be a set of *function symbols*, Π a set of *predicate symbols*, $\mathcal{V}$ a denumerable set of variables and $\mathcal{U} \subseteq \mathcal{V}$. The set $\mathcal{T}_{\Sigma,\mathcal{U}}$ of *terms* over Σ and $\mathcal{U}$ is the smallest set containing x in $\mathcal{U}$ and $f(t_1, \cdots, t_n)$ with $f/n \in \Sigma$, $n \geq 0$ and $t_i \in \mathcal{T}_{\Sigma,\mathcal{U}}$ for $1 \leq i \leq n$. The set $\mathcal{A}_{\Pi,\Sigma,\mathcal{U}}$ of *atoms* over Π and $\mathcal{T}_{\Sigma,\mathcal{U}}$ consists of $p(t_1, \cdots, t_n)$ with $p/n \in \Pi$, $n \geq 0$ and $t_i \in \mathcal{T}_{\Sigma,\mathcal{U}}$ for $1 \leq i \leq n$. Let $vars(O)$ denote the set of variables in O. A substitution is a mapping $\theta : \mathcal{V} \mapsto \mathcal{T}_{\Sigma,\mathcal{V}}$ such that $\{x \in \mathcal{V} \mid x \neq \theta(x)\}$, denoted $dom(\theta)$, is finite. The range of θ is $range(\theta) \stackrel{def}{=} \cup_{X \in dom(\theta)} vars(\theta(X))$. $\theta \restriction \mathcal{U}$ is a substitution such that $(\theta \restriction \mathcal{U})(x) = \theta(x)$ for $x \in \mathcal{U}$ and $(\theta \restriction \mathcal{U})(x) = x$ for $x \notin \mathcal{U}$. A substitution $\theta : \mathcal{V} \mapsto \mathcal{T}_{\Sigma,\mathcal{V}}$ is uniquely extended to a homomorphism $\theta : \mathcal{T}_{\Sigma,\mathcal{V}} \mapsto \mathcal{T}_{\Sigma,\mathcal{V}}$. A renaming substitution is a bijective mapping from $\mathcal{V}$ to $\mathcal{V}$. Let Sub be the set of idempotent substitutions.

An *equation* is a formula $l = r$ where either $l, r \in \mathcal{T}_{\Sigma,\mathcal{V}}$ or $l, r \in \mathcal{A}_{\Pi,\Sigma,\mathcal{V}}$. The set of all equations is denoted Eqn. For a set of equations $E \in \wp(\mathsf{Eqn})$, a unifier of E is a substitution such that $\theta(l) = \theta(r)$ for each $(l = r) \in E$. E is called unifiable if E has a unifier. A unifier θ of E is a most general unifier if for any other unifier σ of E there is a substitution η such that $\sigma = \eta \circ \theta$ where $\circ$ denotes function composition. All most general unifiers of E are equivalent modulo renaming. Let $mgu : \wp(\mathsf{Eqn}) \mapsto Sub \cup \{fail\}$ return either a most general unifier for E if E is unifiable or $fail$ otherwise. $mgu(\{l = r\})$ is also written as $mgu(l, r)$. Let $\theta \circ fail \stackrel{def}{=} fail$ and $fail \circ \theta \stackrel{def}{=} fail$ for any $\theta \in Sub \cup \{fail\}$.

Let VI be the set of variables of interest. VI is usually the set of the variables occurring in the program. We will use a fixed renaming substitution Ψ such that $\Psi(VI) \cap VI = \emptyset$. Ψ is called a tagging substitution in [38].

Abstract interpretation Two semantics of the program are involved in abstract interpretation. One is called concrete and the other abstract. In a compositional definition of semantics, the concrete semantics is defined in terms of a group of semantic functions $f_i : D_i \mapsto E_i$ and the abstract semantics is defined in terms of another group of semantic function $f_i^\sharp : D_i^\sharp \mapsto E_i^\sharp$ such that each abstract semantic function $f_i^\sharp$ simulates its corresponding concrete semantic function f_i. To prove the correctness of the abstract semantics (the program analysis) with respect to the concrete semantics is reduced to proving the correctness of each abstract semantic function $f_i^\sharp$ with respect to its corresponding concrete semantic function f_i. The latter can be done using the Moore family approach [19] when concrete domains D_i and E_i are complete lattices. Let $\gamma_{D_i^\sharp} : D_i^\sharp \mapsto D_i$ and $\gamma_{E_i^\sharp} : E_i^\sharp \mapsto E_i$ be monotonic functions such that $\gamma_{D_i^\sharp}(D_i^\sharp)$ and $\gamma_{E_i^\sharp}(E_i^\sharp)$ are Moore families. Then $f_i^\sharp : D_i^\sharp \mapsto E_i^\sharp$ is correct with respect to $f_i : D_i \mapsto E_i$ iff $f_i(\gamma_{D_i^\sharp}(x^\sharp)) \sqsubseteq_{E_i} \gamma_{E_i^\sharp}(f_i^\sharp(x^\sharp))$ for each $x^\sharp \in D_i^\sharp$.

Abstract interpretation framework The abstract interpretation framework in [31] which we use to present the parametric groundness analysis is based on a concrete semantics of logic programs that is defined in terms of two operators on $(\wp(Sub), \subseteq)$. One is the set union $\cup$ and the other is $UNIFY$ defined as follows. Let $a_1, a_2 \in \mathcal{A}_{\Pi,\Sigma,VI}$ and $\Theta_1, \Theta_2 \in \wp(Sub)$.

$$UNIFY(a_1, \Theta_1, a_2, \Theta_2) = \{unify(a_1, \theta_1, a_2, \theta_2) \neq fail \mid \theta_1 \in \Theta_1 \wedge \theta_2 \in \Theta_2\}$$

where $unify(a_1, \theta_1, a_2, \theta_2) \stackrel{def}{=} mgu(\rho(\theta_1(a_1)), \theta_2(a_2)) \circ \theta_2$ and ρ is a renaming substitution satisfying $(vars(\theta_1) \cup vars(a_1)) \cap (vars(\theta_2) \cup vars(a_2)) = \emptyset$.

Specializing the framework for a program analysis consists in designing an abstract domain $\langle ASub, \sqsubseteq \rangle$, a monotonic function $\gamma_{ASub} : ASub \mapsto \wp(Sub)$ such that $\gamma_{ASub}(ASub)$ is a Moore family and an abstract operator $AUNIFY$ on $\langle ASub, \sqsubseteq \rangle$ such that, for any $a_1, a_2 \in \mathcal{A}_{\Pi,\Sigma,VI}$ and any $\pi_1, \pi_2 \in ASub$,

$$UNIFY(a_1, \gamma_{ASub}(\pi_1), a_2, \gamma_{ASub}(\pi_2)) \subseteq \gamma_{ASub}(AUNIFY(a_1, \pi_1, a_2, \pi_2))$$

since monotonicity of γ_{ASub} implies that $\gamma_{ASub}(\pi_1) \cup \gamma_{ASub}(\pi_2) \subseteq \gamma_{ASub}(\pi_1 \sqcup \pi_2)$ where $\sqcup$ is the least upper bound operator on $\langle ASub, \sqsubseteq \rangle$. Elements of $ASub$

are called abstract substitutions since they describe sets of substitutions. The abstract operator *AUNIFY* is called abstract unification operator as its main functionality is to simulate unification.

4 Non-Parametric Groundness Analysis

This section reformulates the groundness analysis presented in [43] that uses the abstract domain for groundness proposed in [36]. The reformulated groundness analysis will be used in section 5 to obtain the parametric groundness analysis.

4.1 Abstract Domain

A set of substitutions is described by associating each variable in VI with a groundness mode from MO. The abstract domain is thus $\langle \mathsf{Con}, \sqsubseteq_{\mathsf{Con}} \rangle$ where $\mathsf{Con} \stackrel{def}{=} VI \mapsto \mathsf{MO}$ and $\sqsubseteq_{\mathsf{Con}}$ is the pointwise extension of $\trianglelefteq$.[1] $\langle \mathsf{Con}, \sqsubseteq_{\mathsf{Con}} \rangle$ is a complete lattice. The set of substitutions described by an abstract substitution in Con is given by a function $\gamma_{\mathsf{Con}} : \mathsf{Con} \mapsto \wp(\mathsf{Sub})$ defined as follows.

$$\gamma_{\mathsf{Con}}(\theta^\sharp) \stackrel{def}{=} \{\theta \mid \forall X \in VI.((\theta^\sharp(X) = \mathsf{g}) \rightarrow (vars(\theta(X)) = \emptyset)\}$$

γ_{Con} is a monotonic function from $\langle \mathsf{Con}, \sqsubseteq_{\mathsf{Con}} \rangle$ to $\langle \wp(Sub), \subseteq \rangle$. A substitution θ is said to satisfy an abstract substitution $\theta^\sharp$ if $\theta \in \gamma_{\mathsf{Con}}(\theta^\sharp)$.

The abstract unification operator for the non-parametric groundness analysis also deals with groundness of renamed variables. Let $VI^\dagger \stackrel{def}{=} VI \cup \Psi(VI)$. We define $\mathsf{Con}^\dagger \stackrel{def}{=} VI^\dagger \mapsto \mathsf{MO}$ and $\gamma^\dagger_{\mathsf{Con}}(\theta^\sharp) \stackrel{def}{=} \{\theta \mid \forall X \in VI^\dagger.((\theta^\sharp(X) = \mathsf{g}) \rightarrow (vars(\theta(X)) = \emptyset)\}$.

Lemma 1. *$\gamma_{\mathsf{Con}}(\mathsf{Con})$ and $\gamma^\dagger_{\mathsf{Con}}(\mathsf{Con}^\dagger)$ are Moore families.* ∎

4.2 Abstract Unification

Algorithm 1 defines the abstract unification operator $AUNIFY_{\mathsf{Con}}$ for the non-parametric groundness analysis. Given $\theta^\sharp, \sigma^\sharp \in \mathsf{Con}$ and $a_1, a_2 \in \mathcal{A}_{\Pi,\Sigma,VI}$, the renaming substitution Ψ is first applied to a_1 and $\theta^\sharp$ to obtain $\Psi(a_1)$ and $\Psi(\theta^\sharp)$, and $\Psi(\theta^\sharp)$ and $\sigma^\sharp$ are combined to obtain $\zeta^\sharp = \Psi(\theta^\sharp) \cup \sigma^\sharp$. Note that $\zeta^\sharp \in \mathsf{Con}^\dagger$ and a substitution satisfying $\zeta^\sharp$ satisfies both $\Psi(\theta^\sharp)$ and $\sigma^\sharp$. $E_0 = mgu(\Psi(a_1), a_2)$ is then computed. If $E_0 = \mathit{fail}$ then the algorithm returns $\{X \mapsto \mathsf{g} \mid X \in VI\}$ - the infimum of $\langle \mathsf{Con}, \sqsubseteq_{\mathsf{Con}} \rangle$. Otherwise, the algorithm continues. $\eta^\sharp = DOWN_{\mathsf{Con}}(E_0, \zeta^\sharp)$ is then computed. If a variable X occurs in t, (Y/t) in E_0 and Y is ground in $\zeta^\sharp$ then X is ground in $\eta^\sharp$. Then $\beta^\sharp = UP_{\mathsf{Con}}(\eta^\sharp, E_0)$ is computed. If Y/t in E_0 and all variables in t are ground in $\eta^\sharp$ then Y is ground in $\beta^\sharp$. The algorithm finally restricts $\beta^\sharp$ to VI and returns the result.

[1] An element f in Con is represented as $\{x \in VI \mid f(x) = \mathsf{g}\}$ in the literature.

Algorithm 1 *Let* $\theta^\sharp, \sigma^\sharp \in \mathsf{Con}$, $a_1, a_2 \in \mathcal{A}_{\Pi,\Sigma,VI}$.

$$AUNIFY_{\mathsf{Con}}(a_1, \theta^\sharp, a_2, \sigma^\sharp) \stackrel{def}{=} \begin{cases} let \quad E_0 = mgu(\Psi(a_1), a_2) \ in \\ if \quad E_0 \neq fail \\ then \ UP_{\mathsf{Con}}(E_0, DOWN_{\mathsf{Con}}(E_0, \Psi(\theta^\sharp) \cup \sigma^\sharp)) \upharpoonright VI \\ else \ \{X \mapsto \mathsf{g} \mid X \in VI\} \end{cases}$$

$$DOWN_{\mathsf{Con}}(E, \zeta^\sharp) \stackrel{def}{=} \lambda X. \begin{cases} \zeta^\sharp(X), & \textit{if } X \notin range(E) \\ \zeta^\sharp(X) \bigtriangleup (\bigtriangleup_{(Y/t) \in E \wedge X \in vars(t)} \zeta^\sharp(Y)), & \textit{otherwise.} \end{cases}$$

$$UP_{\mathsf{Con}}(E, \eta^\sharp) \stackrel{def}{=} \lambda X. \begin{cases} \eta^\sharp(X), & \textit{if } X \notin dom(E) \\ \eta^\sharp(X) \bigtriangleup (\bigtriangledown_{Y \in vars(E(X))} \zeta^\sharp(Y)), & \textit{otherwise.} \end{cases}$$

The following theorem states the correctness of the non-parametric groundness analysis.

Theorem 1. *For any* $\theta^\sharp, \sigma^\sharp \in \mathsf{Con}$ *and any* $a_1, a_2 \in \mathcal{A}_{\Pi,\Sigma,VI}$,

$$UNIFY(a_1, \gamma_{\mathsf{Con}}(\theta^\sharp), a_2, \gamma_{\mathsf{Con}}(\sigma^\sharp)) \subseteq \gamma_{\mathsf{Con}}(AUNIFY_{\mathsf{Con}}(a_1, \theta^\sharp, a_2, \sigma^\sharp))$$

∎

5 Parametric Groundness Analysis

The input and the output of the parametric groundness analysis by necessity contains a set Para of groundness parameters. They are instantiated after analysis by an assignment of groundness modes to groundness parameters - a function from Para to MO. Therefore, the parametric groundness analysis needs to propagate groundness information encoded by groundness parameters in such a way that instantiating its output by a groundness assignment κ obtains the same groundness information as first instantiating its input by κ and then performing the non-parametric groundness analysis.

5.1 Abstract Domain

We first consider how to describe groundness of a variable in the presence of groundness parameters. In the non-parametric groundness analysis, groundness of a variable is described by a groundness mode from MO. Propagation of groundness reduces to computing the least upper bounds and greatest lower bounds of groundness modes from MO. In the parametric groundness analysis, groundness descriptions of a variable contain parameters and hence the least upper bound and greatest lower bound of groundness descriptions cannot evaluated to an element of MO or an element of Para during analysis. We resolve this problem by delaying the least upper bound and the greatest lower bound computations. This requires that groundness of a variable be described by an expression formed of elements of MO, elements of Para, the least upper bound operator $\bigtriangledown$ and the greatest lower bound operator $\bigtriangleup$. It can be shown that

Observation 1 *Any expression formed as above is equivalent to an expression of the form* $\nabla_{i\in I}(\triangle_{j\in J_i}\alpha_i^j)$ *where* $\alpha_i^j \in \mathsf{Para}$. ∎

Expression $\nabla_{i\in I}(\triangle_{j\in J_i}\alpha_i^j)$ is represented as a set $\mathcal{S}$ of subsets of groundness parameters. Let $\mathcal{S} = \{S_1, S_2, \cdots, S_n\}$, $S_i = \{\alpha_i^1, \alpha_i^2, \cdots, \alpha_i^{k_i}\}$. $\mathcal{S}$ stands for $\nabla_{1\leq i\leq n}(\triangle_{1\leq j\leq k_i}\alpha_i^j)$ which is a function from $(\mathsf{Para} \mapsto \mathsf{MO})$ to MO defined as $\mathcal{S}(\kappa) \stackrel{def}{=} \nabla_{1\leq i\leq n}(\triangle_{1\leq j\leq k_i}\kappa(\alpha_i^j))$. For any $\kappa \in (\mathsf{Para} \mapsto \mathsf{MO})$, $\emptyset(\kappa) = \mathsf{g}$ since $\bigtriangledown\emptyset = \mathsf{g}$ and $\{\emptyset\}(\kappa) = \mathsf{u}$ as $\bigtriangledown(\triangle\emptyset) = \mathsf{u}$. Thus, $\emptyset$ and $\{\emptyset\}$ represent modes g and u respectively. There may be two parametric groundness descriptions $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1(\kappa) = \mathcal{S}_2(\kappa)$ for any $\kappa \in (\mathsf{Para} \mapsto \mathsf{MO})$. We follow the normal practice in program analysis of identifying those descriptions that have the same denotation. Define relations $\ll$ and $\cong$ on $\wp(\wp(\mathsf{Para}))$ as $\mathcal{S}_1 \ll \mathcal{S}_2 \stackrel{def}{=} \forall S_1 \in \mathcal{S}_1.\exists S_2 \in \mathcal{S}_2.(S_2 \subseteq S_1)$ and $\mathcal{S}_1 \cong \mathcal{S}_2 \stackrel{def}{=} (\mathcal{S}_1 \ll \mathcal{S}_2) \wedge (\mathcal{S}_2 \ll \mathcal{S}_1)$. Then $\cong$ is an equivalence relation on $\wp(\wp(\mathsf{Para}))$. The domain of parametric groundness descriptions is $\langle \mathsf{PMO}, \preceq\rangle$ where

$$\mathsf{PMO} \stackrel{def}{=} \wp(\wp(\mathsf{Para}))_{/\cong}$$
$$\preceq \stackrel{def}{=} \ll_{/\cong}$$

$\langle \mathsf{PMO}, \preceq\rangle$ is a complete lattice with its infimum being $[\emptyset]_{\cong}$ and its supremum being $[\{\emptyset\}]_{\cong}$. The least upper bound of $[\mathcal{S}_1]_{\cong}$ and $[\mathcal{S}_2]_{\cong}$ is $[\mathcal{S}_1]_{\cong} \oplus [\mathcal{S}_2]_{\cong} = [\mathcal{S}_1 \cup \mathcal{S}_2]_{\cong}$ and the greatest lower bound of $[\mathcal{S}_1]_{\cong}$ and $[\mathcal{S}_2]_{\cong}$ is $[\mathcal{S}_1]_{\cong} \otimes [\mathcal{S}_2]_{\cong} = [\{S_1 \cup S_2 \mid S_1 \in \mathcal{S}_1 \wedge S_2 \in \mathcal{S}_2\}]_{\cong}$. A parametric groundness description $[\mathcal{S}]_{\cong} \in \mathsf{PMO}$ is a function from $(\mathsf{Para} \mapsto \mathsf{MO})$ to MO defined as $[\mathcal{S}]_{\cong}(\kappa) \stackrel{def}{=} \mathcal{S}(\kappa)$. In other words, a parametric groundness description is instantiated to a non-parametric groundness description - a groundness mode in MO by a groundness assignment.

Let $|X|$ be the number of elements in set X and $size([\mathcal{S}]_{\cong}) \stackrel{def}{=} \Sigma_{S\in\mathcal{S}}|S|$. It can be shown that

Lemma 2. *The height of* PMO *is* $\mathcal{O}(2^{|\mathsf{Para}|})$ *and* $size([\mathcal{S}]_{\cong})$ *is* $\mathcal{O}(|\mathsf{Para}|2^{|\mathsf{Para}|})$ *for any* $[\mathcal{S}]_{\cong} \in \mathsf{PMO}$. ∎

A parametric abstract substitution is a function that maps a variable in VI to a groundness description in PMO. The domain of parametric abstract substitutions is $\langle \mathsf{PCon}, \sqsubseteq_{\mathsf{PCon}}\rangle$ where $\mathsf{PCon} \stackrel{def}{=} VI \mapsto \mathsf{PMO}$ and $\sqsubseteq_{\mathsf{PCon}}$ is the pointwise extension of $\preceq$. $\langle \mathsf{PCon}, \sqsubseteq_{\mathsf{PCon}}\rangle$ is a complete lattice with its infimum being $\{x \mapsto [\emptyset]_{\cong} \mid x \in VI\}$. A parametric abstract substitution $\theta^\sharp \in \mathsf{PCon}$ can be thought of as a function from $(\mathsf{Para} \mapsto \mathsf{MO})$ to Con defined as $\theta^\sharp(\kappa) \stackrel{def}{=} \lambda X \in VI.((\theta^\sharp(X))(\kappa))$, that is, a parametric abstract substitution is instantiated to a non-parametric abstract substitution by a groundness assignment. The meaning of a parametric abstract substitution is given by $\gamma_{\mathsf{PCon}} : \mathsf{PCon} \mapsto ((\mathsf{Para} \mapsto \mathsf{MO}) \mapsto \wp(\mathsf{Sub}))$ defined as follows.

$$\gamma_{\mathsf{PCon}}(\theta^\sharp) \stackrel{def}{=} \lambda\kappa.\{\theta \mid \forall x \in VI.((\theta^\sharp(x)(\kappa) = \mathsf{g}) \rightarrow (vars(\theta(x)) = \emptyset))\}$$

Example 2. Let $\mathsf{Para} = \{\alpha, \beta, \gamma\}$ and $VI = \{x, y, z\}$. $\theta^\sharp = \{x \mapsto [\{\{\alpha, \gamma\}\}]_\cong, y \mapsto [\{\{\beta, \gamma\}\}]_\cong, z \mapsto [\{\{\alpha, \gamma\}, \{\beta, \gamma\}\}]_\cong\}$ is a parametric abstract substitution and $\gamma_{\mathsf{PCon}}(\theta^\sharp)$ is the following function from groundness assignments to sets of substitutions.

$$\left\{\begin{array}{l} \{\alpha \mapsto \mathsf{g}, \beta \mapsto \mathsf{g}, \gamma \mapsto \mathsf{g}\} \mapsto \{\theta \in \mathsf{Sub} \mid vars(\theta(x)) = vars(\theta(y)) = vars(\theta(z)) = \emptyset\} \\ \qquad\qquad\qquad\qquad\vdots \\ \{\alpha \mapsto \mathsf{u}, \beta \mapsto \mathsf{u}, \gamma \mapsto \mathsf{u}\} \mapsto \mathsf{Sub} \end{array}\right\}$$

■

Let $\mathsf{PCon}^\dagger \stackrel{def}{=} VI^\dagger \mapsto \mathsf{PMO}$ and $\gamma^\dagger_{\mathsf{PCon}}(\theta^\sharp) \stackrel{def}{=} \lambda\kappa.\{\theta \mid \forall x \in VI^\dagger.((\theta^\sharp(x)(\kappa) = \mathsf{g}) \rightarrow (vars(\theta(x)) = \emptyset))\}$.

Lemma 3. *$\gamma_{\mathsf{PCon}}(\mathsf{PCon})$ and $\gamma^\dagger_{\mathsf{PCon}}(\mathsf{PCon}^\dagger)$ are Moore families.* ■

5.2 Abstract Unification

Algorithm 2 defines an abstract unification operator for the parametric groundness analysis. It is obtained from that for the non-parametric groundness analysis by replacing non-parametric groundness descriptions with parametric groundness descriptions, ∇ and $\triangle$ by $\oplus$ and $\otimes$ respectively, and renaming $AUNIFY_{\mathsf{Con}}$, $DOWN_{\mathsf{Con}}$ and UP_{Con} into $AUNIFY_{\mathsf{PCon}}$, $DOWN_{\mathsf{PCon}}$ and UP_{PCon} respectively.

Algorithm 2 *Let $\theta^\sharp, \sigma^\sharp \in \mathsf{PCon}$, $a_1, a_2 \in \mathcal{A}_{\Pi,\Sigma,VI}$.*

$$AUNIFY_{\mathsf{PCon}}(a_1, \theta^\sharp, a_2, \sigma^\sharp) \stackrel{def}{=} \left\{\begin{array}{l} let \quad E_0 = mgu(\Psi(a_1), a_2) \ in \\ if \quad E_0 \neq fail \\ then \ UP_{\mathsf{PCon}}(E_0, DOWN_{\mathsf{PCon}}(E_0, \Psi(\theta^\sharp) \cup \sigma^\sharp)) \upharpoonright VI \\ else \ \{X \mapsto [\emptyset]_\cong \mid X \in VI\} \end{array}\right.$$

$$DOWN_{\mathsf{PCon}}(E, \zeta^\sharp) \stackrel{def}{=} \lambda X. \left\{\begin{array}{ll} \zeta^\sharp(X), & if\ X \notin range(E) \\ \zeta^\sharp(X) \otimes (\bigotimes_{(Y/t) \in E \wedge X \in vars(t)} \zeta^\sharp(Y)), & otherwise. \end{array}\right.$$

$$UP_{\mathsf{PCon}}(E, \eta^\sharp) \stackrel{def}{=} \lambda X. \left\{\begin{array}{ll} \eta^\sharp(X), & if\ X \notin dom(E) \\ \eta^\sharp(X) \otimes (\bigoplus_{Y \in vars(E(X))} \zeta^\sharp(Y)), & otherwise. \end{array}\right.$$

Lemma 4. *The time complexity of $AUNIFY_{\mathsf{PCon}}$ is $\mathcal{O}(|VI|^2 |\mathsf{Para}|^2 2^{|\mathsf{Para}|})$ and that of $\sqcup_{\mathsf{PCon}}$ is $\mathcal{O}(|VI||\mathsf{Para}|^2 2^{|\mathsf{Para}|})$ where $\sqcup_{\mathsf{PCon}}$ is the least upper bound operator on PCon.* ■

Example 3. This example illustrates how $AUNIFY_{\mathsf{PCon}}$ works. Let

$$\begin{aligned} VI &= \{X, Y, Z\} \\ A &= g(X, f(Y, f(Z, Z)), Y) \\ B &= g(f(X, Y), Z, X) \\ \theta^\sharp &= \{X \mapsto \{\{\alpha_1, \alpha_2\}\}, Y \mapsto \{\{\alpha_1, \alpha_3\}\}, Z \mapsto \{\{\alpha_2, \alpha_3\}\}\} \\ \sigma^\sharp &= \{X \mapsto \{\{\alpha_1\}, \{\alpha_2\}\}, Y \mapsto \{\{\alpha_2, \alpha_3\}\}, Z \mapsto \{\emptyset\}\} \end{aligned}$$

Suppose $\Psi = \{X \mapsto X_0, Y \mapsto Y_0, Z \mapsto Z_0\}$. We have

$$\begin{array}{ll}\Psi(A) & = g(X_0, f(Y_0, f(Z_0, Z_0)), Y_0) \\ \Psi(\theta^\sharp) & = \{X_0 \mapsto \{\{\alpha_1, \alpha_2\}\}, Y_0 \mapsto \{\{\alpha_1, \alpha_3\}\}, Z_0 \mapsto \{\{\alpha_2, \alpha_3\}\}\} \\ \zeta^\sharp & = \Psi(\theta^\sharp) \cup \sigma^\sharp \\ & = \left\{ \begin{array}{l} X_0 \mapsto \{\{\alpha_1, \alpha_2\}\}, Y_0 \mapsto \{\{\alpha_1, \alpha_3\}\}, Z_0 \mapsto \{\{\alpha_2, \alpha_3\}\}, \\ X \mapsto \{\{\alpha_1\}, \{\alpha_2\}\}, Y \mapsto \{\{\alpha_2, \alpha_3\}\}, Z \mapsto \{\emptyset\} \end{array} \right\}\end{array}$$

and

$$E_0 = eq \circ mgu(\Psi(A), B) = \{X_0 = f(Y_0, Y), Z = f(Y_0, f(Z_0, Z_0)), X = Y_0\}$$

and

$$\begin{array}{ll}\eta^\sharp & = DOWN_{\mathsf{PCon}}(E_0, \zeta^\sharp) \\ & = \left\{ \begin{array}{l} X_0 \mapsto \{\{\alpha_1, \alpha_2\}\}, Y_0 \mapsto \{\{\alpha_1, \alpha_2, \alpha_3\}\}, Z_0 \mapsto \{\{\alpha_2, \alpha_3\}\}, \\ X \mapsto \{\{\alpha_1\}, \{\alpha_2\}\}, Y \mapsto \{\{\alpha_1, \alpha_2, \alpha_3\}\}, Z \mapsto \{\emptyset\} \end{array} \right\} \\ \beta^\sharp & = UP_{\mathsf{PCon}}(E_0, \eta^\sharp) \\ & = \left\{ \begin{array}{l} X_0 \mapsto \{\{\alpha_1, \alpha_2, \alpha_3\}\}, Y_0 \mapsto \{\{\alpha_1, \alpha_2, \alpha_3\}\}, Z_0 \mapsto \{\{\alpha_2, \alpha_3\}\}, \\ X \mapsto \{\{\alpha_1, \alpha_2, \alpha_3\}\}, Y \mapsto \{\{\alpha_1, \alpha_2, \alpha_3\}\}, Z \mapsto \{\{\alpha_2, \alpha_3\} \end{array} \right\}\end{array}$$

Finally,

$$\begin{array}{l} AUNIFY_{\mathsf{PCon}}(A, \theta^\sharp, B, \sigma^\sharp) \\ = \beta^\sharp \restriction VI \\ = \{X \mapsto \{\{\alpha_1, \alpha_2, \alpha_3\}\}, Y \mapsto \{\{\alpha_1, \alpha_2, \alpha_3\}\}, Z \mapsto \{\{\alpha_2, \alpha_3\}\}\} \end{array}$$

∎

The following theorem states that instantiating the output of the parametric groundness analysis by a groundness assignment obtains the same groundness information as first instantiating its input by the same groundness assignment and then performing the non-parametric groundness analysis.

Theorem 2. *Let* $[\mathcal{S}_1]_{\cong}, [\mathcal{S}_2]_{\cong} \in \mathsf{PMO}$, $\theta^\sharp, \sigma^\sharp \in \mathsf{PCon}$, $\eta^\sharp, \zeta^\sharp \in \mathsf{PCon}^\dagger$, $a_1, a_2 \in \mathcal{A}_{\Pi,\Sigma,VI}$ *and* $E \in \wp(\mathsf{Eqn})$. *For any* $\kappa \in (\mathsf{Para} \mapsto \mathsf{MO})$,

(a) $([\mathcal{S}_1]_{\cong} \otimes [\mathcal{S}_2]_{\cong})(\kappa) = (\mathcal{S}_1(\kappa) \triangle \mathcal{S}_2(\kappa))$ *and* $([\mathcal{S}_1]_{\cong} \oplus [\mathcal{S}_2]_{\cong})(\kappa) = (\mathcal{S}_1(\kappa) \nabla \mathcal{S}_2(\kappa))$;
(b) $(DOWN_{\mathsf{PCon}}(E, \zeta^\sharp))(\kappa) = DOWN_{\mathsf{Con}}(E, \zeta^\sharp(\kappa))$;
(c) $(UP_{\mathsf{PCon}}(E, \eta^\sharp))(\kappa) = UP_{\mathsf{Con}}(E, \eta^\sharp(\kappa))$; *and*
(d) $(AUNIFY_{\mathsf{PCon}}(a_1, \theta^\sharp, a_2, \sigma^\sharp))(\kappa) = AUNIFY_{\mathsf{Con}}(a_1, \theta^\sharp(\kappa), a_2, \sigma^\sharp(\kappa))$.

∎

The following theorem establishes the correctness of the parametric groundness analysis.

Theorem 3. *For any* $a_1, a_2 \in \mathcal{A}_{\Pi,\Sigma,VI}$, $\kappa \in (\mathsf{Para} \mapsto \mathsf{MO})$, *and* $\theta^\sharp, \sigma^\sharp \in \mathsf{PCon}$,

$$UNIFY(a_1, \gamma_{\mathsf{PCon}}(\theta^\sharp)(\kappa), a_2, \gamma_{\mathsf{PCon}}(\sigma^\sharp)(\kappa)) \subseteq \gamma_{\mathsf{PCon}}(AUNIFY_{\mathsf{PCon}}(a_1, \theta^\sharp, a_2, \sigma^\sharp))(\kappa)$$

∎

6 Implementation

We have implemented the parametric groundness analysis and the abstract interpretation framework in SWI-Prolog. The abstract interpretation framework is implemented using O'Keefe's least fixed-point algorithm [40]. Both the abstract interpretation framework and the parametric groundness analysis are implemented as meta-interpreters using ground representations for program variables and groundness parameters.

6.1 An Example

Example 4. The following is the permutation sort program from [45] (Chapter 3) and the result of the parametric groundness analysis. The sets are represented by lists. $V \mapsto T$ is written as V/T in the results, α as alpha and β as beta. Program points marked (a) and (b) will be referred to later.

```
:-  %[Li/[[alpha]],Lo/[[beta]]]                                           (a)
  sort(Li,Lo).
    %[Li/[[alpha]],Lo/[[alpha,beta]]]

select(X,[X|Xs],Xs).
    %[X/[[alpha,beta]],Xs/[[alpha]]]
select(X,[Y|Ys],[Y|Zs]) :-
    %[X/[[beta]],Y/[[alpha]],Ys/[[alpha]],Zs/[[]]]
  select(X,Ys,Zs).
    %[X/[[alpha,beta]],Y/[[alpha]],Ys/[[alpha]],Zs/[[alpha]]]

ordered([]).
    %[]
ordered([X]).
    %[X/[[alpha,beta]]]
ordered([X,Y|Ys]) :-
    %[X/[[alpha,beta]],Y/[[alpha,beta]],Ys/[[alpha,beta]]]             (b)
  X=<Y,
    %[X/[],Y/[],Ys/[[alpha,beta]]]
  ordered([Y|Ys]).
    %[X/[],Y/[],Ys/[]]

permutation(Xs,[Z|Zs]) :-
    %[Xs/[[alpha]],Z/[[beta]],Zs/[[beta]],Ys/[[]]]
  select(Z,Xs,Ys),
    %[Xs/[[alpha]],Z/[[alpha,beta]],Zs/[[beta]],Ys/[[alpha]]]
  permutation(Ys,Zs).
    %[Xs/[[alpha]],Z/[[alpha,beta]],Zs/[[alpha,beta]],Ys/[[alpha]]]
permutation([],[])
    %[].

sort(Xs,Ys) :-
    %[Xs/[[alpha]],Ys/[[beta]]]
```

```
permutation(Xs,Ys),
  %[Xs/[[alpha]],Ys/[[alpha,beta]]]
ordered(Ys).
  %[Xs/[[alpha]],Ys/[[alpha,beta]]]
```

The top-level goal is $sort(Li, Lo)$ and the input abstract substitution at program point (a) is $\{Li \mapsto \{\{\alpha\}\}, Lo \mapsto \{\{\beta\}\}\}$. It says that groundness mode of Li is α and that of Lo is β. The parametric groundness analysis infers an abstract substitution for every other program points. The abstract substitution at program point (b) associates the parametric groundness description $\alpha\triangle\beta$ with variables X, Y and Ys. The result can be instantiated by any of four groundness assignments in $\{\alpha, \beta\} \mapsto \mathsf{MO}$. Let $\kappa = \{\alpha \mapsto \mathtt{g}, \beta \mapsto \mathtt{u}\}$. Then κ instantiates the input abstract substitution to $\{Li \mapsto \mathtt{g}, Lo \mapsto \mathtt{u}\}$ and the abstract substitution at program point (b) to $\{X \mapsto \mathtt{g}, Y \mapsto \mathtt{g}, Ys \mapsto \mathtt{g}\}$. This indicates that if the goal $sort(Li, Lo)$ is called with Li being ground then X, Y and Ys are ground when $(X <= Y)$ is invoked.

Since the abstract substitution at program point (b) maps both X and Y to $\alpha\triangle\beta$, it is obvious that if either α or β is assigned $\mathtt{g}$ then X and Y are ground before the execution of $X =< Y$ and the run-time groundness check at program point (b) can be eliminated. ∎

6.2 Performance

The SWI-Prolog implementation of the parametric groundness analysis has been tested with a set of benchmark programs. The experiments were done on an 1.0GHz Dell Desktop running Windows 2000 Professional and SWI-Prolog 3.4.0.

Table 1 shows time performance of the implementation. All but the last row corresponds to a specific input. The input consists of a program, a goal and an input abstract substitution that specifies the groundness of the variables in the goal. The program and the goal are listed in the first and the third columns. The input abstract substitution associates each variable in the goal with a different groundness parameter. For instance, the abstract substitution for the first row is $\{X \mapsto \{\{\alpha\}\}, Y \mapsto \{\{\beta\}\}\}$. The second column lists the size of the program measured in the number of program points in the program. Each fact p is treated as a clause $p \leftarrow true$ which has two program points. The fourth column is the time in seconds spent on the input. The last row gives the total size of the programs and the total time.

Table 1 indicates that the prototype parametric groundness analyzer spends an average of 1.72 seconds to process one thousand program points. This is an acceptable speed for most logic programs. We believe that there is still room for improving the time performance through a better implementation because both meta-programming and ground representation of variables significantly slow the prototype.

The same table compares the performance of the parametric groundness analysis with that of the non-parametric groundness analysis presented in [43] which uses a subset of VI as an abstract substitution. The subset contains those variables that are definitely ground under all substitutions described by the abstract

Program	Points	Goal	Poly (sec)	Mono (sec)	Ratio	Assign-ments
Buggy Quick Sort	38	qs(A, B)	0.038	0.016	2.375	4
Exponentiation	27	exp(A, B, C)	0.01	0.009	1.111	8
Factorial	25	factorial(A, B)	0.008	0.005	1.6	4
Graph Connectivity	50	connected(A, B)	0.012	0.009	1.333	4
Heapify Binary Trees	27	heapify(A, B)	0.043	0.015	2.867	4
Improved Quick Sort	22	iqsort(A, B)	0.025	0.009	2.778	4
Interchange Sort	24	sort(A, B)	0.015	0.005	3	4
List Insertion	23	insert(A, B, C)	0.01	0.008	1.25	8
Permutation Sort	26	sort(A, B)	0.018	0.008	2.25	4
QuickSort with D-List	22	quicksort(A, B)	0.027	0.012	2.25	4
Tree Sort	34	treesort(A, B)	0.038	0.014	2.714	4
ann	653	go(A)	0.911	0.541	1.684	2
asm	904	asm_PIL(A, B)	1.188	0.855	1.389	4
boyer	351	tautology(A)	0.269	0.16	1.681	2
browse	132	q	0.073	0.06	1.217	1
chat	1368	chat_parser	4.326	2.554	1.694	1
cs_r	348	pgenconfig(A)	0.649	0.42	1.545	2
disj_r	180	top(A)	0.103	0.088	1.17	2
dnf	95	go	0.285	0.239	1.192	1
ga	503	test_ga	0.541	0.531	1.019	1
gabriel	131	main(A, B)	0.122	0.072	1.694	4
kalah	298	play(A, B)	0.215	0.144	1.493	4
life	115	lift(A, B, C, D)	0.04	0.04	1	16
mastermind	238	play	0.13	0.1	1.3	1
meta	110	interpret(A)	0.139	0.073	1.904	2
nand	624	main(A)	1.117	0.921	1.213	2
naughts_and_crosses	137	play(A)	0.067	0.042	1.595	2
nbody	454	go(A, B)	0.404	0.235	1.719	4
neural	382	test(A, B)	0.257	0.119	2.16	4
peep	541	comppeepopt(A, B, C)	1.07	0.538	1.989	8
press	455	test_press(A, B)	1.624	0.626	2.594	4
queens	33	queens(A, B)	0.01	0.007	1.429	4
read	500	read(A, B)	1.863	1.219	1.528	4
reducer	408	try(A, B)	0.719	0.426	1.688	4
ronp	110	puzzle(A)	0.097	0.05	1.94	2
sdda	355	do_sdda(test, A, B, C)	0.462	0.23	2.009	8
semi	216	go(A, B)	0.773	0.382	2.024	4
serialize	50	go(A)	0.083	0.033	2.515	2
simple_analyzer	560	main(A)	0.716	0.39	1.836	2
tictactoe	286	play(A)	0.34	0.263	1.293	2
tree order	39	v2t(A, B, C)	0.038	0.021	1.81	8
tsp	153	tsp(A, B, C, D, E)	0.164	0.087	1.885	32
zebra	64	zebra(A, B, C, D, E, F, G)	0.048	0.025	1.92	128
	11111		19.087		1.78	7.4

Table 1. Performance of Parametric Groundness Analysis

substitution. This allows operators on abstract substitutions to be optimized. The non-parametric groundness analysis is implemented in the same way as the parametric groundness analysis.

The number of different groundness assignments for the parametric groundness analysis is two to the power of the number of groundness parameters. Each assignment corresponds to a non-parametric groundness analysis that is performed and measured. The fifth column lists the average time in seconds spent on these non-parametric groundness analyses. The sixth column lists the ratio of the fourth column and the fifth column. The seventh column lists the number of groundness assignments.

The table shows that the time the parametric groundness analysis takes is from 1.0 to 3.0 times that the non-parametric groundness analysis takes. On average, the parametric groundness analysis is 78% slower. This is due to the fact that the parametric groundness descriptions are more complex than the non-parametric groundness descriptions. The abstract unification operator and the least upper bound operator for the parametric groundness are more costly than those for the non-parametric groundness analysis.

The result of the parametric groundness analysis is much more general than that of the non-parametric groundness analysis. It can be instantiated as many times as there are different groundness assignments. The average number of groundness assignments is 7.4 which is 4.2 times the average performance ratio 1.78. In order to derive a sufficient condition for safely removing groundness checks for builtin calls, the non-parametric groundness analysis must be run as many times as the number of groundness assignments. In this case, the parametric groundness analysis is 4.2 times better.

7 Related Work

The parametric groundness analysis has been obtained from a non-parametric groundness analysis that uses a simple groundness domain. As groundness is useful both in compile-time program optimizations and in improving the precision of other program analyses, more powerful groundness domains have been studied. These domains consists of propositional formulae over program variables that act as propositional variables. Dart uses the domain Def of definite propositional formulae to capture groundness dependency between variables [22]. For instance, the definite propositional formula $x \leftarrow (y \wedge z)$ represents the groundness dependency that x is bound to a ground term if y and z are bound to ground terms. Def consists of propositional formulae whose models are closed under set intersection [17]. Marriott and Søndergaard use the domain Pos (also called Prop) of positive propositional formulae [34]. A propositional formula f is positive if f is true when all propositional variables in f are true. Pos is strictly more powerful than Def. It has been further studied in [17, 2, 3] and has several implementations [29, 25, 8, 4].

Giacobazzi and Scozzari reconstruct Def and Pos from Con via Heyting completion [23, 42] where Con is a subdomain of Pos and consists of propositional formulae that are conjunctions of propositional variables. The Sharing domain proposed by Jacobs and Langen for sharing analysis [26, 27] also contains groundness information. Cortesi et. al prove that groundness information contained in Sharing is exactly that captured by Def [16, 15]. Codish and Søndergaard recently discover that Sharing is isomorphic to Pos in structure [10].

Pos-based analyzers using binary decision diagrams have been shown [25] to be precise and efficient for benchmark programs. However, Pos-based analyzers do not come with any efficiency guarantee as they require in the worst case exponential number of iterations or exponentially large data structures [11]. More abstract domains [22, 12, 24] have been proposed, offering different trade-offs between the precision and the efficiency of analysis.

Pos-based goal-independent groundness analyzers enjoys a favorable property of being condensing [27, 34]. An analysis F that infers output information $F(P, \phi)$ from a program P and input information ϕ is condensing if $F(P, \phi \wedge \psi) = F(P, \phi) \wedge \psi$ for any P, ϕ and ψ. Thus, a condensing analysis can be performed with partial input information ϕ and its output be conjoined with additional input information ψ to obtain the output that would result from analyzing the program with complete input information $\phi \wedge \psi$. Thus, a Pos-based goal-independent groundness analysis is also parametric since its result can be instantiated by logic conjunction. [34] and [8] present two approaches to perform condensing goal-independent groundness analysis using program transformation and bottom-up evaluation. An atomic call in the transformed program in [34] contains both variables of interest at a program point in the original program and variables in the query. Thus, the abstract domain for goal-independent groundness analysis in [34] is $\mathsf{Pos}_{(\mathsf{Para} \cup VI)}$ since variables in the query play the role of groundness parameters. Similar argument can be made of [8].

Groundness analyzers in [25, 17] use Pos with top-down abstract interpretation frameworks to perform goal-dependent groundness analysis [25, 17]. These analyzers project a Pos formula onto variables occurring in the clause to which the program point belongs. This makes them fail to capture groundness dependency between variables at a program point and variables in a query. Let Pos_X denote the set of positive Boolean functions over X - the set of propositional variables. The following fix should make a top-down Pos-based groundness analysis condensing and hence parametric. The abstract domain Pos_{VI} is extended to $\mathsf{Pos}_{(\mathsf{Para} \cup VI)}$ and the projection operation $\lambda f.\bar{\exists}_{VI}.f$ is replaced with $\lambda f.\bar{\exists}_{(\mathsf{Para} \cup VI)}.f$.

Though Pos-based groundness analysis is parametric and more precise than the parametric groundness analysis, the parametric groundness analysis is more efficient. The cost of an analysis is determined by the number of iterations performed and cost of operations performed in each iteration. The height of $\mathsf{Pos}_{(\mathsf{Para} \cup VI)}$ (abstract domain in a Pos-based goal-dependent groundness analysis) is $\mathcal{O}(2^{|VI|} 2^{|\mathsf{Para}|})$ [3]. The height of PCon (abstract domain in the parametric groundness analysis) is $\mathcal{O}(|VI| 2^{|\mathsf{Para}|})$. Therefore, the number of iterations performed in the parametric groundness analysis is much less than those performed in a Pos-based groundness analysis. Abstract operations $AUNIFY_{\mathsf{PCon}}$ and $\sqcup_{\mathsf{PCon}}$

are also much less expensive than $\vee$, $\wedge$ and existential quantification on positive Boolean functions over $(\mathsf{Para} \cup VI)$ in a Pos-based groundness analysis. Therefore, the parametric groundness analysis is more efficient than a Pos-based groundness analysis. Furthermore, the parametric groundness analysis has the same asymptotic time and space complexity as a Con-based groundness analysis when it is used with $\mathsf{Para} = \emptyset$. Thus, it only pays more cost than a Con-based analysis when it infers more general results.

$$\begin{array}{l} append([\,], L, L). \\ append([H|L_1], L_2, [H|L_3]) \leftarrow append(L_1, L_2, L_3).ⓑ \end{array}$$

$$\leftarrow ⓐ\ \mathrm{append}(Xs, Ys, Zs).ⓒ$$

Fig. 2. The append program

It is interesting to note that the parametric groundness analysis also captures some groundness dependency among variables.

Example 5. For the program and the goal in Figure 2, the parametric groundness analysis infers

$$\begin{array}{l} ⓐ : \{Xs \mapsto \{\{\alpha\}\}, Ys \mapsto \{\{\beta\}\}, Zs \mapsto \{\{\gamma\}\}\} \\ \Longrightarrow ⓒ : \{Xs \mapsto \{\{\alpha, \gamma\}\}, Ys \mapsto \{\{\beta, \gamma\}\}, Zs \mapsto \{\{\alpha, \gamma\}, \{\beta, \gamma\}\}\} \end{array}$$

This implies that whenever Xs and Ys are bound to ground terms at point ⓒ, Zs is bound to a ground term at the same point. In order to bind Xs to a ground term, g must be assigned to either α or γ. In order to bind Ys to a ground term, g must be assigned to either β or γ. Any groundness assignment satisfying the above two conditions will evaluate $\{\{\alpha, \gamma\}, \{\beta, \gamma\}\}$ to g. So, we have $Xs \wedge Ys \rightarrow Zs$ in Pos. Similarly, we can infer $Xs \wedge Ys \leftarrow Zs$. ∎

In general, if the abstract substitution at a program point assign $\mathcal{R}_j$ to Y_j for $1 \leq j \leq l$ and $\mathcal{S}_i$ to X_i for $1 \leq i \leq k$ and $\oplus_{1 \leq j \leq l} \mathcal{R}_j \preceq \oplus_{1 \leq i \leq k} \mathcal{S}_k$ then the Pos like proposition $\wedge_{1 \leq i \leq k} X_i \rightarrow \wedge_{1 \leq j \leq l} Y_j$ holds at the program point. Thus the parametric groundness analysis also captures groundness dependency between program variables. However, the degree to which the parametric groundness analysis captures this kind of groundness dependency is limited. In particular, when $\mathsf{Para} = \emptyset$, the parametric groundness analysis degenerates to the non-parametric groundness analysis which does not capture this kind of groundness dependency.

There have also been effort in analyzing logic programs to discover type dependency between program variables [7, 32, 13]. Though groundness modes in MO can be thought of as types, it is not beneficial to apply a type dependency analysis to infer groundness dependency. Abstract domains for type dependency analyses in [7, 32, 13] are more complex and hence abstract operations are more costly than those required in a groundness dependency analysis. Furthermore, their abstract domains have infinite increasing chains and they must employ a widening operator. [32] obtains a parametric type analysis from a non-parametric type analysis. Since types have much rich structures than groundness modes, equational constraints over parametric types need be incorporated into its abstract domain in order to propagate precisely type dependency. This makes abstract operations costly. As there are infinite number of assignments of types to type parameters, loss of precision is incurred when abstract operations in the parametric type analysis mimicks those in the non-parametric type analysis. The parametric groundness analysis presented in this paper has a much simpler abstract domain and abstract operations that mimicks precisely those in the non-parametric groundness analysis.

8 Conclusion

We have presented a new groundness analysis, called parametric groundness analysis, that infers groundness of variables parameterized by groundness parameters that can be instantiated after analysis. The parametric groundness analysis is obtained by generalizing a non-parametric groundness analysis. Experimental results with a prototype implementation of the analysis are promising. The parametric groundness analysis is as precise as the non-parametric groundness analysis.

The parametric groundness analysis is theoretically faster but less precise than a Pos based groundness analysis. As future work, we would like to compare experimentally the time and the precision of the parametric groundness analysis with those of Pos based groundness analyses.

References

1. K.R. Apt. Logic programming. In J.V. Leeuwen, editor, *Handbook of Theoretical Computer Science: (Volume B) Formal Models and Semantics*, pages 493–574. Elsevier Science Publishers B.V., 1990.
2. T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Boolean functions for dependency analysis: Algebraic properties and efficient representation. *Lecture Notes in Computer Science*, 864:266–280, 1994.
3. 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.
4. R. Bagnara and P. Schachte. Factorizing equivalent variable pairs in ROBDD-based implementations of *Pos*. Lecture Notes in Computer Science, 1548:471–485, 1999.
5. M. Bruynooghe. A practical framework for the abstract interpretation of logic progams. *Journal of Logic Programming*, 10(2):91–124, 1991.
6. M. Codish, D. Dams, and Yardeni E. Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. In K. Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming*, pages 79–93. The MIT Press, 1991.

7. M. Codish and B. Demoen. Deriving polymorphic type dependencies for logic programs using multiple incarnations of Prop. *Lecture Notes in Computer Science*, 864:281–297, 1994.
8. M. Codish and B. Demoen. Analysing logic programs using "Prop"-ositional logic programs and a magic wand. *Journal of Logic Programming*, 25(3):249–274, 1995.
9. M. Codish, A. Mulkers, M. Bruynooghe, M. García de la Banda, and M. Hermenegildo. Improving abstract interpretations by combining domains. In PEPM93 [41], pages 194–205.
10. M. Codish and H. Søndergaard. The boolean logic of set sharing analysis. *Lecture Notes in Computer Science*, 1490:89–101, 1998.
11. Michael Codish. Worst-case groundness analysis using positive boolean functions. *The Journal of Logic Programming*, 41(1):125–128, 1999.
12. Michael Codish, Andy Heaton, Andy King, Muhamed Abo-Zaed, and Pat Hill. Widening Positive Boolean functions for Goal-dependent Groundness Analysis. Technical Report 12-98, Computing Laboratory, May 1998.
13. Michael Codish and Vitaly Lagoon. Type dependencies for logic programs using aci-unification. *Journal of Theoretical Computer Science*, 238:131–159, 2000.
14. A. Cortesi and G. Filé. Abstract interpretation of logic programs: an abstract domain for groundness, sharing, freeness and compoundness analysis. In *Proceedings of the Symposium on Partial Evaluation and Semantics-based Program Manipulation*, pages 52–61, New Haven, Connecticut, USA, 1991.
15. A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. *ACM Transactions on Programming Languages and Systems*, 19(1):7–47, 1997.
16. A. Cortesi, G. Filé, and W. Winsborough. Comparison of abstract interpretations. In *Proceedings of the 19th Int. Colloquium on Automata, Languages and Programming ICALP'92*, pages 523–534. Springer Verlag, 1992.
17. A. Cortesi, G. Filé, and W. Winsborough. Optimal groundness analysis using propositional logic. *Journal of Logic Programming*, 27(2):137–168, 1996.
18. P. Cousot and R. Cousot. Abstract interpretation: a unified framework for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the fourth annual ACM symposium on Principles of programming languages*, pages 238–252. The ACM Press, 1977.
19. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the sixth annual ACM symposium on Principles of programming languages*, pages 269–282. The ACM Press, 1979.
20. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(1, 2, 3 and 4):103–179, 1992.
21. M. A. Covington, D. Nute, and A. Vellino. *PROLOG PROGRAMMING IN DEPTH*. Scott, Foresman & Co., 1988.
22. P.W. Dart. On derived dependencies and connected databases. *Journal of Logic Programming*, 11(2):163–188, 1991.
23. R. Giacobazzi and F. Scozzari. Intuitionistic Implication in Abstract Interpretation. In *Proceedings of Programming Languages: Implementations, Logics and Programs*, pages 175–189. Springer-Verlag, 1997. LNCS 1292.
24. Andy Heaton, Muhamed Abo-Zaed, Michael Codish, and Andy King. Simple, efficient and scalable groundness analysis of logic programs. *The Journal of Logic Programming*, 45(1-3):143–156, 2000.
25. P. Van Hentenryck, A. Cortesi, and B. Le Charlier. Evaluation of the Domain `PROP`. *Journal of Logic Programming*, 23(3):237–278, 1995.

26. D. Jacobs and A. Langen. Accurate and Efficient Approximation of Variable Aliasing in Logic Programs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 154–165, Cleveland, Ohio, USA, 1989.
27. D. Jacobs and A. Langen. Static analysis of logic programs for independent and parallelism. *Journal of Logic Programming*, 13(1–4):291–314, 1992.
28. T. Kanamori. Abstract interpretation based on Alexander Templates. *Journal of Logic Programming*, 15(1 & 2):31–54, 1993.
29. B. Le Charlier and P. Van Hentenryck. Groundness analysis of Prolog: implementation and evaluation of the domain Prop. In PEPM93 [41], pages 99–110.
30. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
31. L. Lu. Abstract interpretation, bug detection and bug diagnosis in normal logic programs. PhD thesis, University of Birmingham, 1994.
32. L. Lu. A polymorphic type analysis in logic programs by abstract interpretation. *Journal of Logic Programming*, 36(1):1–54, 1998.
33. K. Marriott and H. Søndergaard. Bottom-up dataflow analysis of normal logic programs. *Journal of Logic Programming*, 13(1–4):181–204, 1992.
34. 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.
35. K. Marriott, H. Søndergaard, and N.D. Jones. Denotational abstract interpretation of logic programs. *ACM Transactions on Programming Languages and Systems*, 16(3):607–648, 1994.
36. C. Mellish. Some global optimisations for a Prolog compiler. *Journal of Logic Programming*, 2(1):43–66, 1985.
37. C. Mellish. Abstract interpretation of Prolog programs. In S. Abramsky and C. Hankin, editors, *Abstract interpretation of declarative languages*, pages 181–198. Ellis Horwood Limited, 1987.
38. K. Muthukumar and M. Hermenegildo. Compile-time derivation of variable dependency using abstract interpretation. *Journal of Logic Programming*, 13(1, 2, 3 and 4):315–347, 1992.
39. U. Nilsson. Towards a framework for the abstract interpretation of logic programs. In *Proceedings of the International Workshop on Programming Language Implementation and Logic Programming*, pages 68–82. Springer-Verlag, 1988.
40. R. A. O'Keefe. Finite fixed-point problems. In J.-L. Lassez, editor, *Proceedings of the fourth International Conference on Logic programming*, volume 2, pages 729–743. The MIT Press, 1987.
41. *ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*. The ACM Press, 1993.
42. F. Scozzari. Logical optimality of groundness analysis. *Lecture Notes in Computer Science*, 1302:83–97, 1997.
43. H. Søndergaard. An application of abstract interpretation of logic programs: occur check problem. *Lecture Notes in Computer Science*, 213:324–338, 1986.
44. H. Søndergaard. Immediate fixpoints and their use in groundness analysis. *Lecture Notes in Computer Science*, 1180:359–370, 1996.
45. L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, 1986.
46. R. Sundararajan and J.S. Conery. An abstract interpretation scheme for groundness, freeness, and sharing analysis of logic programs. In R. Shyamasundar, editor, *Proceedings of 12th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 203–216. Springer-Verlag, 1992.