Towards Parametrizing Logic Program Analysis: Two Examples (Technical Communication)

Lunjin Lu
Oakland University

submitted 10 April 2013; accepted 23 May 2013

Abstract

A parametric analysis is an analysis whose input and output are parametrized with a number of parameters which can be instantiated to abstract properties after analysis is completed. This paper proposes to use Cousot and Cousot's Cardinal power domain to capture dependencies of analysis output on its input and obtain a parametric analysis by parametrizing a base analysis. We illustrate the method by parametrizing $\mathcal{P}os$-based groundness dependency and set sharing analyses of logic programs. Experiments with a prototype analyzer shows that generality of the two resulting parametric analyses comes with a small extra cost.

1 Introduction

A program analysis infers information from programs. Let P be a program, $\mathcal{I}$ express input information, and $\mathcal{O}$ express output information inferred from P and $\mathcal{I}$. We write $\langle \mathcal{I}, P, \mathcal{O} \rangle$ to denote the analysis that infers $\mathcal{O}$ from P and $\mathcal{I}$. A typical program analysis is non-parametric in the sense that the program need be analyzed separately for different input information. Note that program variables are not parameters for input information, though input information is usually a predicate over program variables. Take the generic sorting program $sort(x, y)$ for instance, letting nat denote the set of natural numbers, int the set of integers, and $list(\beta)$ the set of lists of elements from β, program analyses $\langle x \in list(nat), sort(x, y), y \in list(nat) \rangle$ and $\langle x \in list(int), sort(x, y), y \in list(int) \rangle$ are accomplished separately even if they are two instances of a parametric analysis $\langle x \in list(\beta), sort(x, y), y \in list(\beta) \rangle$ where both input information and output information are parametrized. By assigning different values to β which serves as a place holder for information to be filled in after analysis, $\langle x \in list(\beta), sort(x, y), y \in list(\beta) \rangle$ can be instantiated into many different non-parametric analyses such as $\langle x \in list(nat), sort(x, y), y \in list(nat) \rangle$ and $\langle x \in list(int), sort(x, y), y \in list(int) \rangle$. Parametric program analyses infer more general results, which brings some benefits. Firstly, a sub-program or a library program need not be analyzed separately for its different uses, i.e., the result of a parametric analysis is reusable. This has positive bearing on analysis efficiency because output information for different uses of the same sub-program can be obtained by instantiation rather than by re-analysis. Secondly, parametric analyses are amenable to program modifications. A sub-program need not be re-analyzed if it does not invoke a modified sub-program directly or indirectly. In addition, its analysis result is re-instantiated only if it is directly invoked by a modified sub-program.

The goal of this paper is to design a method that lifts a base analysis to a parametric analysis that is correct and as precise as its base analysis: if $\langle \mathcal{I}(\vec{\beta}), P, \mathcal{O}(\vec{\beta})\rangle$ is the result of the parametric analysis then $\langle \mathcal{I}(\kappa), P, \mathcal{O}(\kappa)\rangle$ is the result of the base analysis for any possible value κ for $\vec{\beta}$. Observe that both $\mathcal{I}(\vec{\beta})$ and $\mathcal{O}(\vec{\beta})$ are functions from the domain of the values for parameters to the domain of properties for the base analysis.

The contributions of the paper are as follows. Firstly, a systematic approach is presented for deriving a parametric analysis from a given base analysis. This involves lifting the semantic domain for the base analysis to its Cardinal power with respect to the domain of parameter values and lifting the semantic function accordingly. Secondly, this approach is applied to two goal-dependent analyses for logic programs: Søndergaard and Marriott's groundness (dependency) analysis and Codish, Søndergaard and Stuckey's (set) sharing analysis. Both are parameterized with parameters that express groundness of the variables in the top-level goal.

The next section provides background knowledge on abstract interpretation. Section 3 describes the approach to parametrizing program analyses and section 4 applies the approach to goal-dependent groundness and sharing analyses for logic programs and briefly describes experimental results with a prototype implementation of these parametric analyses. Section 5 concludes. Proofs are omitted.

2 Preliminaries

This section recalls some concepts and results in abstract interpretation (Cousot and Cousot 1979; Cousot and Cousot 1992a) that are used in the remainder of the paper. A semantics of a program is given by an interpretation $\langle (D, \sqsubseteq_D), f\rangle$ where $(D, \sqsubseteq_D)$ is a complete lattice and f is a monotone function on $(D, \sqsubseteq_D)$. The semantics is defined as the least fixed point lfp f of f. The concrete semantics of the program is given by the concrete interpretation $\langle (D, \sqsubseteq_D), f\rangle$ while an abstract semantics is given by an abstract interpretation $\langle (D^\sharp, \sqsubseteq_D^\sharp), f^\sharp\rangle$. The correspondence between the concrete domain $(D, \sqsubseteq_D)$ and the abstract domain $(D^\sharp, \sqsubseteq_D^\sharp)$ is formalized by a Galois connection (α, γ). The function $\alpha : D \mapsto D^\sharp$ is called an abstraction function and the function $\gamma : D^\sharp \mapsto D$ a concretization function. They are monotone functions such that $\forall c \in D.(c \sqsubseteq_D \gamma \circ \alpha(c))$ and $\forall a \in D^\sharp.(\alpha \circ \gamma(a) \sqsubseteq_D^\sharp a)$ hold. A sufficient condition for lfp$f^\sharp$ to be a safe abstraction of lfp f is $\forall a \in D^\sharp.(\alpha \circ f \circ \gamma(a) \ \sqsubseteq_D^\sharp \ f^\sharp(a))$ or equivalently $\forall a \in D^\sharp.(f \circ \gamma(a) \ \sqsubseteq_D \ \gamma \circ f^\sharp(a))$, according to propositions 24 and 25 in (Cousot and Cousot 1992a). In a compositional design of analysis, both f and $f^\sharp$ are defined in terms of primitive semantic functions such that each primitive abstract semantic function $f_i^\sharp : D_i^\sharp \mapsto E_i^\sharp$ simulates its corresponding primitive concrete semantic function $f_i : D_i \mapsto E_i$. To prove the correctness of the abstract semantics 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. Let $\gamma_{D_i^\sharp} : D_i^\sharp \mapsto D_i$ and $\gamma_{E_i^\sharp} : E_i^\sharp \mapsto E_i$ be concretization functions. 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 $x^\sharp \in D_i^\sharp$.

3 Parametrizing Program Analyses

A parametric analysis is obtained from a base analysis by lifting its primitive abstract domains and operations.

3.1 Lifting Semantic Domains to Cardinal Power Domains

The cardinal power $L_1 \xrightarrow{m} L_2$ with base L_2 and exponent L_1 consists of all monotone functions from L_1 to L_2. We parametrize a base analysis by lifting its abstract domain to its Cardinal power using the domain over which analysis parameters range as the exponent abstract domain. The following proposition from (Cousot and Cousot 1992b) states that there is a Galois connection between the abstract and concrete Cardinal power domains and it is derived from the Galois connection between concrete and abstract base domains and the Galois connection between the concrete and abstract exponent domains.

Proposition 3.1
(Cousot and Cousot 1992b) Let $\langle L_1, \alpha_1, L_1^\sharp, \gamma_1\rangle$ and $\langle L_2, \alpha_2, L_2^\sharp, \gamma_2\rangle$ be Galois connections. Then $\langle L_1 \xrightarrow{m} L_2, \alpha, L_1^\sharp \xrightarrow{m} L_2^\sharp, \gamma\rangle$ is a Galois connection where $\alpha = \lambda\phi.\alpha_2 \circ \phi \circ \gamma_1$ and $\gamma = \lambda\psi.\gamma_2 \circ \psi \circ \alpha_1$.

3.2 Lifting Semantic Functions

The domain of an interpretation is often formed from a number of primitive domains and the semantic function from a number of primitive functions between primitive domains. We now define a family of operators $\star_L$ that lift a monotone function $f : D \xrightarrow{m} E$ to a monotone function $\star_L f : (L \xrightarrow{m} D) \xrightarrow{m} (L \xrightarrow{m} E)$.

Definition 3.1
Let $f : D \xrightarrow{m} E$. Define $\star_L f : (L \xrightarrow{m} D) \xrightarrow{m} (L \xrightarrow{m} E)$ as

$$\star_L f = \lambda\phi.(f \circ \phi)$$

In this defintion, D and E are primitive (concrete or abstract) domains and f a transfer function from D to E. $\star_L f$ is a transfer function from $(L \xrightarrow{m} D)$ to $L \xrightarrow{m} E$. Note that D can be a Cartesian product of n sets with $n \geqslant 0$. Each element e in E can be identified as a constant function in $D \xrightarrow{m} E$ that maps each $d \in D$ to e. Then $\star_L e = \lambda\phi.e$.

The following theorem shows that lifting operators commute with composition, tupling and projection, indicating that lifting of the semantic function of an interpretation can be accomplished by lifting its primitive semantic functions.

Theorem 3.2
For any L,

1. $\star_L(f_2 \circ f_1) = (\star_L f_2) \circ (\star_L f_1)$ for any $f_1 : D \xrightarrow{m} E$ and $f_2 : E \xrightarrow{m} F$,
2. $\star_L\langle f_1, f_2\rangle = \langle \star_L f_1, \star_L f_2\rangle$ for any $f_1 : D \xrightarrow{m} E$ and $f_2 : D \xrightarrow{m} F$,
3. $\star_L proj_i(\langle \phi_1, \phi_2\rangle) = \phi_i$ for $i = 1, 2$, $\phi_1 : L \xrightarrow{m} D_1$ and $\phi_2 : L \xrightarrow{m} D_2$ where $proj_i(\langle c_1, c_2\rangle) = c_i$ for $i = 1, 2$.

The following theorem states that performing the parametric analysis with a parametrized input and then binding the parameters to abstract properties yields the same result as the base analysis performed with the instantiation of the input with the same binding. Thus, the parametric analysis is as precise as the base analysis.

Theorem 3.3
Let $f : D \xrightarrow{m} D$ and $\kappa : L \xrightarrow{m} D$. Then, for any $\ell \in L$,

$$\mathrm{lfp}_{\kappa(\ell)}\, f = (\mathrm{lfp}_{\kappa}\, (\star_L f))(\ell)$$

Remark 3.4
In fact, any fixpoint of $\star_L f$ provides a set of fixpoints of f. Let $f : D \xrightarrow{m} D$ and $\kappa : L \xrightarrow{m} D$ such that $\kappa = (\star_L f)(\kappa)$. Then, for any $\ell \in L$, $\mathrm{lfp}_{\kappa(\ell)}\, f = \kappa(\ell)$ since $f(\kappa(\ell)) = f \circ \kappa(\ell) = ((\star_L f)(\kappa))(\ell) = \kappa(\ell)$.

Let $\langle L_2, \alpha_2, L_2^\sharp, \gamma_2 \rangle$ be a Galois connection and $f : L_2 \xrightarrow{m} L_2$ and $f^\sharp : L_2^\sharp \xrightarrow{m} L_2^\sharp$ the concrete and abstract semantic functions. The concrete and abstract domains L_2 and $L_2^\sharp$ can be parametrized by L_1 and $L_1^\sharp$ which are related to each other by a Galois connection $\langle L_1, \alpha_1, L_1^\sharp, \gamma_1 \rangle$. The following theorem says that correctness of the base analysis implies correctness of the parametric analysis and optimality of the base analysis implies optimality of the parametric analysis.

Theorem 3.5
Let $\langle L_1, \alpha_1, L_1^\sharp, \gamma_1 \rangle$ and $\langle L_2, \alpha_2, L_2^\sharp, \gamma_2 \rangle$ be Galois connections, $f : L_2 \xrightarrow{m} L_2$ and $f^\sharp : L_2^\sharp \xrightarrow{m} L_2^\sharp$. Let α and γ be defined as in Proposition 3.1. Then

1. If $\alpha_2 \circ f \circ \gamma_2 \sqsubseteq f^\sharp$, $\alpha \circ (\star_{L_1} f) \circ \gamma \sqsubseteq \star_{L_1^\sharp} f^\sharp$.
2. If $\alpha_2 \circ f \circ \gamma_2 = f^\sharp$ and $\langle L_1, \alpha_1, L_1^\sharp, \gamma_1 \rangle$ is a Galois insertion, $\alpha \circ (\star_{L_1} f) \circ \gamma = \star_{L_1^\sharp} f^\sharp$.

The abstract domains $L_1^\sharp$ and $L_2^\sharp$ are in general abstractions of different concrete domains. Elements in $L_1^\sharp$ are properties of initial program states whilst elements in $L_2^\sharp$ properties of reachable program states at a given program point. Cardinal power $L_1^\sharp \xrightarrow{m} L_2^\sharp$ captures dependencies of the properties of reachable program states on those of initial program states.

Analysis Input for Patametric Analysis For the initial program point where program execution starts, $L_1^\sharp$ and $L_2^\sharp$ are abstractions of the same concrete domain – $L_1 = L_2$. Assume that the set of all possible initial program states be described by ℓ_2 in $L_2^\sharp$ which may be derived from the specification of the program. For each ℓ_1 in $L_1^\sharp$, there is an abstract property $\alpha_2(\gamma_1(\ell_1))$ in $L_2^\sharp$ that most accurately describes all those concrete elements that are described by ℓ_1. Hence, each ℓ_1 corresponds to an analysis input to the base analysis $\ell_2 \sqcap_{L_2^\sharp} \alpha_2(\gamma_1(\ell_1))$ where $\sqcap_{L_2^\sharp}$ is the meet operator on $L_2^\sharp$. Therefore, analysis input for parametric analysis is $\iota_{\ell_2} = \lambda \ell_1.(\ell_2 \sqcap_{L_2^\sharp} \alpha_2 \circ \gamma_1(\ell_1))$.

4 Groundness and Sharing Analyses

The above approach obtains a parametric analysis from the abstract domain and operations for a base analysis and an exponent abstract domain. Theorems 3.3 and 3.5 guarantee the correctness and preciseness of the parametric analysis. We now briefly describe the primitive abstract domains and operations for Marriott and Søndergaard's groundness analysis (Marriott and Søndergaard 1993) and those for Codish, Søndergaard and Stuckey's sharing analysis (Codish et al. 1999). Both of these analyses use positive

Boolean formulas to describe properties of substitutions which are program states in logic programming. These analyses will be paramertized and have been chosen for two reasons. Firstly, groundness and sharing are well studied properties for logic programs. Secondly, these two analyses share the same abstract domains and three of five abstract operations, which simplifies presentation. We first describe the domain of positive Boolean formulas.

4.1 Positive Boolean Formulas

Let U be a finite set of Boolean variables. A Boolean formula over U is formed of Boolean constants $\mathbf{0}$ and $\mathbf{1}$, Boolean variables from U and logical connectives $\wedge$, $\vee$, $\leftrightarrow$ and $\neg$. Other connectives such as $\rightarrow$ and $\leftarrow$ can be defined using these connectives. Let $Bool = \{\mathbf{0}, \mathbf{1}\}$ ordered by $\mathbf{0} \leqslant \mathbf{1}$. A truth substitution μ on U is a partial function from U to $Bool$. The application of μ to a Boolean formula b is denoted $\mu(b)$. Let $\mu = \{x \mapsto \mathbf{1}\}$ and $b = (x \rightarrow y)$. Then $\mu(b) = (\mathbf{1} \rightarrow y)$. If μ is defined for every Boolean variables in b then μ is called a truth assignment for b. Given a formula b and a truth assignment μ, we write $\mu \models b$ and call μ a model of b if $\mu(b)$ evaluates to $\mathbf{1}$. $b_1 \models b_2$ means that $\mu \models b_1$ implies $\mu \models b_2$ for every truth assignment μ for b_1. As is customary (see e.g. (Codish et al. 1999)), a model μ of a Boolean formula b over U is used interchangeably with $\{x \in U \mid \mu(x) = \mathbf{1}\}$ – the set of those variables in U that are mapped to $\mathbf{1}$ by μ. Two formulas b_1 and b_2 are equivalent, denoted $b_1 = b_2$ if both $b_1 \models b_2$ and $b_2 \models b_1$. We shall not distinguish between elements in an equivalence class of $=$. A Boolean formula b is positive if $\mathbf{u} \models b$ for each such truth substitution $\mathbf{u}$ that assigns $\mathbf{1}$ to all the Boolean variables in b. Let $\mathcal{P}os_U$ denote the set of positive Boolean formulas over U. Then $\langle \mathcal{P}os_U, \models \rangle$ is a complete lattice with bottom $\wedge U$, top $\mathbf{1}$, meet $\wedge$ and join $\vee$ where $\wedge$ and $\vee$ are respectively logical conjunction and disjunction. A positive formula is called definite if the set of its models (viewed as sets) is closed under set intersection (Dart 1991). We use $\mathcal{D}ef_U$ to denote the set of definite Boolean formulas over U. The subscript U will be dropped from $\mathcal{P}os_U$ and $\mathcal{D}ef_U$ when either it is clear from the context or it does not matter.

4.2 Groundness Analysis

In logic programming, a value is a term that may contain variables. In any program state during the execution of a logic program, logic variables are bound to terms that may contain variables which are bound to other terms later during execution. A variable x is ground in a substitution θ if θ maps x to a term that does not contain any variable. Groundness is one of the most studied properties for logic programs (Codish and Demoen 1995; Cortesi et al. 1996; Dart 1991; Marriott and Søndergaard 1993; Scozzari 2002). Marriott and Søndergaard (Marriott and Søndergaard 1993) proposed to use positive Boolean formulas to capture groundness dependencies between variables in a program state. The formula x describes those program states in which x is bound to a ground term while $x \rightarrow y$ describes those program states in which y is ground whenever x is. The analysis uses a family of abstract domains $\langle \mathcal{P}os_U, \models \rangle$ - one for each set of variables of interest U because different sets of variables are of interest at different program points. Some notations are needed to present abstract operations for the groundness anlaysis. A sequences of different variables is denoted by a bold low case letter. The dimension of $\vec{x}$ is denoted $|\vec{x}|$. Let $\vec{x} = x_1 x_2 \cdots x_m$ and $\vec{y} = y_1 y_2 \cdots y_m$. We write $\vec{x} \leftrightarrow \vec{y}$ as a shorthand for $\bigwedge_{1 \leqslant i \leqslant m}(x_i \leftrightarrow y_i)$. We sometimes write $\vec{x}$ for $\{x_i \mid 1 \leqslant i \leqslant m\}$ to simplify notation when such a use is clear from the context. Let $X = \{x_1, x_2, \cdots x_m\}$. We write $\exists X.\phi$ as a shorthand for $\exists x_1.\exists x_2.\cdots \exists x_m.\phi$. The set of variables occuring in a syntactic object o is denoted $\mathbf{V}(o)$. The following are the abstract operations for Marriott and Søndergaard's groundness analysis.

- Abstract join operation $\sqcup^{Gr}$ is logical disjunction operation $\vee$.
- Abstract renaming operation $rename^{Gr}_{\vec{x} \mapsto \vec{y}}$ is defined for all $\vec{x}$ and $\vec{y}$ such that $|\vec{x}| = |\vec{y}|$. $rename^{Gr}_{\vec{x} \mapsto \vec{y}}(\phi) = \exists \vec{x}.(\phi \wedge (\vec{x} \leftrightarrow \vec{y}))$ if $\mathbf{V}(\phi) \cap \vec{y} = \emptyset$. Otherwise, $rename^{Gr}_{\vec{x} \mapsto \vec{y}}(\phi) = rename^{Gr}_{\vec{z} \mapsto \vec{y}} \circ rename^{Gr}_{\vec{x} \mapsto \vec{z}}(\phi)$ where $\vec{z}$ is such that $(\mathbf{V}(\phi) \cup \vec{x} \cup \vec{y}) \cap \vec{z} = \emptyset$. For instance, $rename^{Gr}_{x_1 x_2 \mapsto x_2 x_1}(x_1 \rightarrow x_2) = (x_2 \rightarrow x_1)$.
- Abstract projection operation $project^{Gr}_X$ is defined $project^{Gr}_X(\phi) = \exists X.\phi$.
- Abstract identity operation $ident^{Gr}_U$ is defined $ident^{Gr}_U = \mathbf{1}$. Note that $\mathbf{1}$ is the only Boolean formula that describes the identity substitution which maps each variable to itself.
- Abstract unification operation $aunify^{Gr}_{x=t}$ is defined $aunify^{Gr}_{x=t}(\phi) = \phi \wedge (x \leftrightarrow \bigwedge \mathbf{V}(t))$ where x is a variable and t a term.

The correctness of these operations is well established.

4.3 Sharing Analysis

Sharing domains track possible sharing between program variables since optimisations and transformations can typically only be applied in the absence of sharing. Sharing is as well studied as groundness (see e.g., (Bagnara et al. 2000; Codish et al. 1999; Cortesi and Filé 1999; Jacobs and Langen 1992; Muthukumar and Hermenegildo 1991)). For a given set of variables of interests U, the Jacob and Langen's sharing domain JL consists of those sets $\mathcal{S}$ of subsets of U such that $\mathcal{S}$ contains $\emptyset$. Each set G in a JL abstraction $\mathcal{S}$ indicates that it is possible to further instantiate those and only those variables in G by binding a single variable. Absence of a variable x from all sets in $\mathcal{S}$ implies that x is ground in all substitutions that are described by $\mathcal{S}$. Groundness information captured by the JL domain is exactly that captured by $\mathcal{D}ef$ (Cortesi et al. 1998).

Codish, Søndergaard and Stuckey encode JL abstractions over U as positive Boolean formulas over U and provide operations on positive Boolean formulas that emulate those on JL abstractions. Let $\mathcal{S}$ be encoded as ϕ. Then each G in $\mathcal{S}$ corresponds to a model μ of ϕ in that $\mu = U \setminus G$. The sharing domain JL over U is isomorphic to $\mathcal{P}os_U$ (Codish et al. 1999). Thus, Codish etc.'s sharing domain over U is also $\langle \mathcal{P}os_U, \models \rangle$. It is important to note that while the same domain is used for both the groundness and sharing analyses, a Boolean formula has different interpretations in these analyses. Abstract join $\sqcup^{Sh}$, renaming $rename^{Sh}_{\vec{x} \mapsto \vec{y}}$ and projection $project^{Sh}_X$ operations for Codish etc.'s sharing analysis have the same definitions as their counterparts for Marriott etc.'s groundness analysis. The following are abstract identity and unification operations for Codish etc.'s sharing analysis.

- $ident^{Sh}_U = \bigwedge U \vee \bigvee_{x \in U}(\neg x \wedge \bigwedge(U \setminus \{x\}))$. E.g., $ident^{Sh}_{\{x_1,x_2\}} = x_1 \vee x_2$.
- $aunify^{Sh}_{x=t}(\phi) = \phi \wedge (x \vee d) \vee H(\phi \wedge (\neg x \vee \neg d)) \wedge (\neg x \wedge \neg d)$ where $d = \bigwedge \mathbf{V}(t)$ and H is a upper closure operator that takes a $\mathcal{P}os$ formula ψ to the smallest (with respect to $\models$) $\mathcal{D}ef$ formula ψ' such that $\psi \models \psi'$ (Schachte and Søndergaard 2006).

The above definitions for $ident^{Sh}_{U}$ and $aunify^{Sh}_{x=t}$ are equivalent to those in (Codish et al. 1999). Correctness of $rename^{Sh}_{\vec{x}\mapsto\vec{y}}$ can be proved the same way as that of $rename^{Gr}_{\vec{x}\mapsto\vec{y}}$; and correctness of other abstract operations is established in (Codish et al. 1999).

4.4 Parametrizing Groundness and Sharing Analyses

A parametric analysis informs about how the abstract property at a program point depends on that at an initial program point. Let $\mathbb{P}$ be a set of parameters and each element of $\mathbb{P}$ stands for the groundness of a variable in the top-level goal before execution. We parameterize the groundness and sharing analyses by the simplest groundness domain $\mathcal{G}_{\mathbb{P}}$ proposed by Jones and Søndergaard (Jones and Søndergaard 1987). $\mathcal{G}_{\mathbb{P}}$ captures groundness information in a substitution in terms of the collection of the variables that are grounded by the substitution; it is the set of conjunctive Boolean formulae with Boolean variables from $\mathbb{P}$

$$\mathcal{G}_{\mathbb{P}} \quad = \quad \{\wedge X \mid X \subseteq \mathbb{P}\}$$

ordered by logical implication $\models$.

The primitive abstract domains of the parametric groundness and sharing analyses obtained by lifting to Cardinal powers with exponent $\mathcal{G}_{\mathbb{P}}$ are $\mathcal{G}_{\mathbb{P}} \overset{m}{\mapsto} \mathcal{P}os_{U}$. Applying Definition 3.1, we obtain abstract operations for the parametric groundness analysis: $\phi_1\ (\star\sqcup^{Gr})\ \phi_2 = \lambda g.(\phi_1(g)\sqcup^{Gr}\phi_2(g))$, $\star project^{Gr}_{X}(\phi) = project^{Gr}_{X} \circ \phi$, $\star rename^{Gr}_{\vec{x}\mapsto\vec{y}}(\phi) = rename^{Gr}_{\vec{x}\mapsto\vec{y}} \circ \phi$, $\star ident^{Gr}_{U} = \lambda g.ident^{Gr}_{U}$, and $(\star aunify^{Gr}_{x=t})(\phi) = aunify^{Gr}_{x=t} \circ \phi$ where the subscript $\mathcal{G}_{\mathbb{P}}$ in $\star_{\mathcal{G}_{\mathbb{P}}}$ has been dropped. Abstract operations for the parametric sharing analysis may be obtained by changing the superscript Gr in these definitions to Sh.

4.5 Prototype Implementation

We have implemented a logic program analyzer in SICStus Prolog and the CUDD package that can perform base and parametric groundness and sharing analyses. The analyzer has been tested with a suite of benchmark programs. The experiments were done on a 2.33GHz Intel (R) Xeon (R) CPU running Linux 2.6.24 and SICSTUS Prolog 4.0.3. The CUDD package version is 2.4.1. The results from the experiments indicate that the prototype analyzer spends 0.135 (resp. 0.296) seconds per one thousand atoms in the abstract program to perform parametric groundness (resp. sharing) analysis. This is an acceptable speed for most logic programs. They also show that the time the parametric groundness analysis takes is from 0.88 to 1.18 times that the base groundness analysis takes with an average of 1.023. For sharing analysis, the time the parametric analysis takes is from 0.97 to 1.74 with an average of 1.087. This indicates that extra cost is small for performaing the parametric groundness and set sharing analyses which yield more general and instantiable results, which is quite surprising and promising. Note that a single run of a parametric analysis infers information that is derived by 2^n runs of its base analysis where n is the number of variables in the top-level goal.

5 Conclusion

We have proposed an approach to parametrizing a base analysis by systematically lifting its primitive abstract domains to their cardinal powers and lifting its abstract operations accordingly. The approach has been applied to parametrize $\mathcal{P}os$-based groundness and sharing analyses for logic programs with a simple groundness domain $\mathcal{G}_{\mathbb{P}}$. Experiments with a prototpye analyzer on a suite of benchmark programs show that the parametric groundness analysis is almost as fast as its base analysis and that the parametric sharing analysis is only slightly slower than its base analysis.

References

Bagnara, R., Zaffanella, E., and Hill, P. 2000. Enhanced sharing analysis techniques: A comprehensive evaluation. In *Proceedings of the Second International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, M. Gabbrielli and F. Pfenning, Eds. ACM Press, 103–114.

Codish, M. and Demoen, B. 1995. Analysing logic programs using “Prop”-ositional logic programs and a magic wand. *Journal of Logic Programming 25,* 3, 249–274.

Codish, M., Søndergaard, H., and Stuckey, P. 1999. Sharing and groundness dependencies in logic programs. *ACM Transactions on Programming Languages and Systems 21,* 5, 948–976.

Cortesi, A. and Filé, G. 1999. Sharing is optimal. *Journal of Logic Programming 38,* 3, 371–386.

Cortesi, A., Filé, G., and Winsborough, W. 1996. Optimal groundness analysis using propositional logic. *Journal of Logic Programming 27,* 2, 137–168.

Cortesi, A., Filé, G., and Winsborough, W. 1998. The quotient of an abstract interpretation. *Theoretical Computer Science 202,* 1–2, 163–192.

Cousot, P. and Cousot, R. 1979. Systematic design of program analysis frameworks. In *Principles of Programming Languages*. The ACM Press, 269–282.

Cousot, P. and Cousot, R. 1992a. Abstract interpretation and application to logic programs. *Journal of Logic Programming 13,* 1, 2, 3 and 4, 103–179.

Cousot, P. and Cousot, R. 1992b. Abstract interpretation frameworks. *J. Logic and Comput. 2,* 4, 511–547.

Dart, P. 1991. On derived dependencies and connected databases. *Journal of Logic Programming 11,* 2, 163–188.

Jacobs, D. and Langen, A. 1992. Static analysis of logic programs for independent and parallelism. *Journal of Logic Programming 13,* 1–4, 291–314.

Jones, N. D. and Søndergaard, H. 1987. A semantics-based framework for the abstract interpretation of Prolog. In *Abstract Interpretation of Declarative Languages*, S. Abramsky and C. Hankin, Eds. Ellis Horwood Ltd, 123–142.

Marriott, K. and Søndergaard, H. 1993. Precise and Efficient Groundness Analysis for Logic Programs. *ACM Letters on Programming Languages and Systems 2,* 4, 181–196.

Muthukumar, K. and Hermenegildo, M. 1991. Combined determination of sharing and freeness of program variables through abstract interpretation. In *Proceedings of the Eighth International Conference on Logic Programming*, K. Furukawa, Ed. The MIT Press, 49–63.

Schachte, P. and Søndergaard, H. 2006. Closure operators for robdds. In *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings*, E. A. Emerson and K. S. Namjoshi, Eds. Lecture Notes in Computer Science, vol. 3855. Springer, 1–16.

Scozzari, F. 2002. Logical optimality of groundness analysis. *Theor. Comput. Sci. 277,* 1-2, 149–184.