Understanding Finiteness Analysis Using Abstract Interpretation (Extended Abstract)[1]

Peter A. Bigot
Saumya Debray
Department of Computer Science
The University of Arizona
Tucson, AZ 85721
{pab, debray}@cs.arizona.edu

Kim Marriott
IBM T.J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598
kimbal@watson.ibm.com

Abstract

Finiteness analyses are compile-time techniques to determine (sufficient) conditions for the finiteness of relations computed during the bottom-up execution of a logic program. We examine finiteness analyses from the perspective of abstract interpretation. However, problems arise when trying to use standard abstract interpretation theory for finiteness analysis. They occur because finiteness is not an *admissible* property and so naive application of abstract interpretation leads to incorrect analyses. Here we develop three simple techniques based on abstract interpretation theory which allow inadmissible properties to be handled. Existing approaches to finiteness analysis may be explained and compared in terms of our extension to abstract interpretation theory, and we claim that their correctness is more easily argued in it. To support our claim we use our techniques to develop and prove correct a finiteness analysis which is more accurate than any that we are aware of.

1 Introduction

Recently a great deal of attention has been devoted to the study of bottom-up execution models for logic programs in which programs are executed by evaluating entire relations and then "joining" them. In implementations of such systems, it is necessary to ensure that the relations computed during any evaluation are finite. In this paper, we are concerned with compile-time analysis techniques to determine (sufficient) conditions for the finiteness of relations computed during the bottom-up execution of a logic program. We examine finiteness analyses from the perspective of abstract interpretation [2, 3]. Abstract interpretation provides a simple semantic basis for dataflow analysis of logic programs. Early work in this area focussed on top-down execution models, as exemplified by Prolog (e.g., see [5, 13, 19]); more recently, researchers have considered abstracting from the bottom-up T_P semantics [11, 12]. However, problems arise when trying to use standard abstract interpretation theory for finiteness analysis. They occur because finiteness is not an *admissible* property (i.e., the union of a chain of finite sets may not be finite), and so naive application of abstract interpretation leads to incorrect analyses. This is because Scott-induction,

[1] The second author was supported in part by the National Science Foundation under grant number CCR-8901283.

the cornerstone of proving correctness of an abstract interpretation, is not valid for inadmissible properties.

Here we develop three simple techniques based on abstract interpretation theory which allow inadmissible properties to be handled. For each technique we illustrate its use in finiteness analysis. The first technique is based on the idea of not developing an analysis for the inadmissible property itself, but rather an analysis for an admissible property which implies the inadmissible. The second technique is to approximate the greatest fixpoint, rather than the least. The third approach is to combine the first two, giving rise to an approach somewhat akin to the widening/ narrowing technique of Cousot & Cousot [3, 4].

This paper has two main technical contributions. The first is the development of generic methods for developing and understanding the dataflow analysis of inadmissible program properties. These methods are interesting theoretically as an extension of abstract interpretation theory and are also important practically as some pragmatically interesting program properties, such as finiteness or fairness, are inadmissible. The second main technical contribution is a framework for the better understanding and development of finiteness analysis. Existing approaches to finiteness analysis may be explained and compared in terms of our extension to abstract interpretation theory, and we claim that their correctness is more easily argued in it. To support our claim we use our techniques to develop and prove correct a finiteness analysis which is more accurate than any that we are aware of.

2 Preliminaries

2.1 Basic Definitions

We assume that the reader is familiar with the basic concepts of latice theory. A subset Y of a complete partial order (cpo) X is a *lower set* iff $\forall y \in Y . \forall x \in X . x \leq y \Rightarrow x \in Y$. *Upper set* is defined dually. A lower set is a *principal ideal* iff it has a maximum element and an upper set is a *principal filter* iff it has a minimum element. A function F is *monotonic* iff for all $x, x' \in X$, $x \leq x' \Rightarrow F\ x \preceq F\ x'$. Dually F is *co-monotonic* iff $x \leq x' \Rightarrow F\ x' \preceq F\ x$). Unless stated to the contrary, all functions will be assumed to be monotonic. The powers of a function $F : X \to X$ on a complete lattice X are defined by

$$F{\uparrow}_x^{\beta} = \begin{cases} x \sqcup F\ (F{\uparrow}_x^{(\beta-1)}) & \text{if } \beta \text{ is a successor ordinal} \\ x \sqcup \bigsqcup \{F{\uparrow}_x^{\beta'} \mid \beta' < \beta\} & \text{otherwise.} \end{cases}$$

$$F{\downarrow}_x^{\beta} = \begin{cases} x \sqcap F\ (F{\downarrow}_x^{(\beta-1)}) & \text{if } \beta \text{ is a successor ordinal} \\ x \sqcap \bigsqcap \{F{\downarrow}_x^{\beta'} \mid \beta' < \beta\} & \text{otherwise.} \end{cases}$$

We let $F{\uparrow}^{\beta}$ denote $F{\uparrow}_{\perp}^{\beta}$ and $F{\downarrow}^{\beta}$ denote $F{\downarrow}_{\top}^{\beta}$. Note that by definition $F{\uparrow}_x^0 = F{\downarrow}_x^0 = x$. Let X be a cpo and $F : X \to X$ be monotonic. It is well known that F has a *least fixpoint*, denoted by $lfp\ F$, such that $lfp\ F = F{\uparrow}^{\beta}$ for some ordinal β; if X is a complete lattice then F also has a *greatest fixpoint*, denoted by $gfp\ F$, such that $gfp\ F = F{\downarrow}^{\beta}$ for some ordinal β. We call the sequence $F{\uparrow}^0, F{\uparrow}^1, ..., F{\uparrow}^{\beta}$ the *Kleene sequence* of F.

2.2 Abstract Interpretation

Abstract interpretation provides the basis for a semantic approach to the development of dataflow analyzers. Abstract interpretation formalizes the idea of "approximate computation" in which computation is performed with descriptions of data rather than the data itself. Correctness of the analysis with respect to the standard interpretation is argued by providing an "approximation relation" which holds whenever an element in a non-standard domain describes an element in the corresponding standard domain. Equivalently, an approximation relation may be defined in terms of a "concretization function" which maps elements in a non-standard domain to those elements in the standard domain which they describe, or an "abstraction function" which maps standard elements to the elements which describe them. Note that these definitions are non-standard because both the concretization and abstraction function map to sets of objects rather than a single object. This generalization is necessary because in the standard framework it is impossible to express an inadmissible approximation relation. The approximation relation can be viewed as a *logical relation* and so it can be lifted from the "base" or "primitive" domains to functions.

Definition. A *description* $Desc = \langle D, \propto, E\rangle$ consists of a *description domain* (a complete lattice) D, a *data domain* (a cpo) E, and an *approximation relation* $\propto \subseteq D \times E$. The function $\gamma : D \to \wp\, E$ defined by $(\gamma\ d) = \{e \mid d \propto e\}$ is called the *concretization function*, and the function $\alpha : E \to \wp\, D$ defined by $(\alpha\ e) = \{d \,|\, d \propto e\}$ is called the *abstraction function*. We require γ to be monotonic and α to be co-monotonic. Let $\langle D_1, \propto_1, E_1\rangle$ and $\langle D_2, \propto_2, E_2\rangle$ be descriptions, and $F : D_1 \to D_2$ and $F' : E_1 \to E_2$ be functions. Then the approximation relation is lifted to functions by defining $F \propto F'$ iff $\forall d \in D_1 .\ \forall e \in E_1 .\ d \propto_1 e \Rightarrow F(d) \propto_2 F'(e)$. By an abuse of notation we will let *Desc* denote both the description and the description domain. ■

Definition. A function $F : X \to \wp\, Y$ is *downwards closed* iff for all $x \in X$, $F(x)$ is a lower set. Dually define upwards closed. ■

It follows from Lemma 4.7 in [10] that:

Lemma 2.1 The abstraction and concretization functions associated with a description are respectively upwards and downwards closed. ■

In this paper our example descriptions will have the desirable property that every object and function has a best or most precise description. This occurs when the description is "singular".

Definition. A description $\langle D, \propto, E\rangle$ is *singular* iff the associated abstraction function $\alpha : E \to \wp\, D$ maps to principal filters. In this case we define an associated *inducement function* $\widehat{\alpha} : E \to D$ by $\widehat{\alpha}\ e = \sqcap(\alpha\ e)$.

We lift the inducement function to function spaces as follows. Let $\langle D_1, \propto_1, E_1\rangle$ be a description, $\langle D_2, \propto_2, E_2\rangle$ a singular description with inducement function $\widehat{\alpha}_2$, and $F : E_1 \to E_2$ a function. We define $\widehat{\alpha}\ F$ to be the function $F' : D_1 \to D_2$ given by

$$F'\ d = \bigsqcup\{\widehat{\alpha}_2\ (F\ e) \mid d \propto_1 e\}.$$

We say that F' is the function *induced* from F. ■

In traditional abstract interpretation additional requirements are placed on descriptions so as to ensure that the approximation relation is admissible. Common restrictions to ensure admissibility are that the abstraction function maps to principal filters and is continuous as in Nielson [16], or that the concretization function maps to principal ideals as in Marriott & Søndergaard [12], or the combination of these two restrictions yielding the Galois connection approach of Cousot & Cousot [2, 3]. See Marriott [10] for more details on the relationships between these restrictions.

Definition. A predicate Q on cpo D is *admissible* iff for all chains $C \subseteq D$, $(\forall c \in C \,.\, Q\ c) \Rightarrow Q\ (\sqcup C)$. Dually, Q is *co-admissible* iff for all chains $C \subseteq D$, $(\forall c \in C \,.\, Q\ c) \Rightarrow Q\ (\sqcap C))$. A description is *(co-)admissible* iff its approximation relation is (co-)admissible. ■

The reason for this is so that Scott induction can be used to prove:

Proposition 2.2 Let D be an admissible description for E and let $F_D : D \rightarrow D$ approximate $F_E : E \rightarrow E$. Then $lfp\ F_D \propto lfp\ F_E$. ■

This result is the basis for proving an abstract interpretation is correct. However the proposition does not hold if the admissibility requirement is dropped, and so when the approximation relation is inadmissible there is no guarantee that an analysis developed using the traditional techniques of abstract interpretation will be correct.

However, even though admissibility of the approximation relation is an implicit requirement of existing abstract interpretation theory, approximation relations that are not admissible do occur in practice. These arise whenever we are interested in finding a property P of a program or database which is itself not admissible, as any natural description domain for the analysis will contain an element which describes exactly those objects with property P, and the approximation relation for such descriptions will therefore be inadmissible. As we shall see, finiteness is an example of such a property.

2.3 Finiteness Analysis

The language we consider is that of Horn logic. We assume that there is some fixed set of *intensional* predicate symbols, IP, and some disjoint set of *extensional* predicate symbols, EP. Given a set of predicate symbols S, HB_S denotes the set of ground atoms that can be constructed from S. Note that we allow an infinite number of constant symbols. In the case S is the set of all predicate symbols we simply write HB. A *program* P is a set of definite clauses (or rules) in which only intensional predicates appear in the heads of the clauses. A *relation* for predicate set S is a subset of HB_S. The *projection* of relation R onto predicate Q, written R_Q, is the set $\{Q(\vec{t}) \mid Q(\vec{t}) \in R\}$. The *projection* of relation R onto argument i, written R_i, is the set $\{\vec{t}_i \mid Q(\vec{t}) \in R\}$ where $\vec{t}_i$ denotes the i^{th} element of the tuple $\vec{t}$. We will only project relations onto arguments after projecting the relation onto a predicate.

The *input* to a program P is a relation for EP. This is called the *extensional database* (EDB). The extensional predicates may have *integrity constraints* associated with them, in which case the input relation is required to satisfy such constraints. The *output* of P is a relation for all of the predicates. The output restricted to the intensional predicates is called the *intensional database* (IDB). The output is computed as the least fixpoint of the "immediate consequence" operator T_P^E where

E is the EDB. The operator $T_P^E : \wp\, HB \rightarrow \wp\, HB$ is defined by:

$$T_P^E(R) = E \cup \left\{ A \;\middle|\; \begin{array}{l} A \;:-\; A_1, \ldots, A_n \text{ is a ground instance of a clause in } P \text{, and} \\ \{A_1, \ldots, A_n\} \subseteq R \end{array} \right\}$$

The *finiteness analysis problem* for a program P, intensional predicate Q, and integrity constraints $\mathcal{E}$ is to determine whether the relation for Q in $lfp(T_P^E)$ is guaranteed to be finite for every set of input relations E satisfying the integrity constraints.

Finiteness analysis is clearly decidable for Datalog programs, i.e., programs where all function symbols have arity 0, if the EDB relations are always finite. The problem becomes undecidable if programs contain function symbols with nonzero arity. The latter problem can be simplified somewhat by approximating the effect of each function symbol of arity $n > 0$ using an infinite EDB relation of arity $n+1$ with certain associated finiteness constraints. The language where programs contain no function symbol of nonzero arity, but where EDB relations may sometimes be infinite, is referred to as *extended Datalog*. As far as we know, the finiteness analysis problem for extended Datalog is not known to be either decidable or undecidable (see, for example, [18]). In this paper, we primarily consider extended Datalog programs. However the analyses are still correct in the presence of arbitrary function symbols.

Before presenting analyses for finiteness, we must define the basic domains which comprise the descriptions on which the analyses are based and how integrity constraints are specified. The idea is to describe each R_Q by a propositional formula describing the finiteness dependencies between the arguments of predicate Q. For instance, the description $p(x, y) : x \rightarrow y$ is read as "for any finite assignment of values to the first argument of predicate p, there are only finitely many assignments to the second argument which satisfy the relation assigned to p".

Definition. Let $Prop$ denote the subset of propositional formula constructed from the constants $true$ and $false$, and a suitably large but finite propositional variable set using the propositional connectives $\wedge, \vee, \leftrightarrow, \rightarrow$.

A *predicate description* for predicate Q has the form $Q(\vec{x}) : \phi$ where $\phi \in Prop$ and $vars(\phi) \subseteq vars(\vec{x})$. Predicate descriptions for predicate Q are pre-ordered by $\leq$ where $(Q(\vec{x}) : \phi) \leq (Q(\vec{x'}) : \phi')$ iff $\phi \Rightarrow \phi''$ where $(Q(\vec{x}) : \phi'')$ is a variable renaming of $(Q(\vec{x'}) : \phi')$.

A *relation description* is a set of predicate descriptions in which a predicate occurs at most once. If there is no predicate description for Q in relation description $\Re$ we implicitly include the description $Q(\vec{x}) : false$. We define $\Re_Q$ to be the predicate description of Q in relation description $\Re$. We let $RDesc$ denote the set of relation descriptions. Relation descriptions are pre-ordered by $\leq$ where $\Re \leq \Re'$ iff $\forall Q \,.\, \Re_Q \leq \Re'_Q$. ■

By an abuse of notation we will treat relation and predicate descriptions modulo the equivalence induced by $\leq$ and order them by the partial order induced from $\leq$. Thus $RDesc$ is a finite lattice.

Relation descriptions will be used both to specify integrity constraints and as descriptions in our finiteness analyses. Usually, we will only allow relation descriptions without disjunction to specify the integrity constraints involving finiteness for the EDB.

Definition. Relation R for predicate Q *satisfies* predicate description $Q(\vec{x}) : \phi$ iff

$$\forall R' \subseteq R \,.\, R' \neq \emptyset \Rightarrow \tau_{R'} \; satisfies \; \phi$$

where the truth assignment $\tau_{R'}$ to the variables $\vec{x}$ is given by $(\tau_{R'}\ \vec{x}_i)$ iff R'_i is finite. Relation R *satisfies* relation description $\Re$, written $R \vdash \Re$ (or $\Re \dashv R$), iff for all predicates Q, R_Q satisfies $\Re_Q$.

The *finiteness description* $FDesc$ is the description $\langle RDesc, \dashv, \wp\, HB\rangle$. ■

Other descriptions for finiteness analysis may be naturally obtained by restricting the type of propositional formula in the predicate descriptions. Natural restrictions are to disallow disjunction, to disallow implication, or to disallow both.

Example 2.1 Consider the infinite relation p representing the immediate successor relation. Given a finite restriction on the first argument position, the set of satisfying tuples has only finitely many values appearing in the second argument position and vice versa. Thus p satisfies $p(x, y) : x \leftrightarrow y$. As another example, the only relation for q to satisfy $q(x) : false$ is the empty relation.

Lemma 2.3 $FDesc$ is singular. ■

Ignoring admissibility can lead to the naive development of an incorrect dataflow analysis.

Definition. Let P be a program. Define $U_P : RDesc \to RDesc \to RDesc$ to be the function induced from $T_P : \wp\, HB \to \wp\, HB \to \wp\, HB$ using the description $FDesc$. That is, for an EDB $E \in \wp\, HB$ described by $\mathcal{E} \in RDesc$, $U_P^{\mathcal{E}}\ F = \sqcup\{\widehat{\dashv}(T_P^E S) \mid \Re \vdash R \wedge \mathcal{E} \vdash E\}$. ■

In more concrete terms, soundness of the propositional logic and the decomposition of relations induced by P permits us to show that:

$$U_P^{\mathcal{E}}\ F = \mathcal{E} \cup \{p(\vec{x}) \colon \phi \mid \phi = \bigvee_{p(\vec{x})\ :-\ q_{i,1}(\vec{v}_{i,1}),\ldots,q_{i,n_i}(\vec{v}_{i,n_i}) \in P} \exists \vec{y}_i \colon \psi_i\}$$

where $\psi_i = \bigwedge_{q_{i,j}(\vec{v}_{i,j}) : \phi_{i,j} \in_\alpha F} \phi_{i,j}$ and $\vec{y}_i = \mathrm{vars}(\psi_i) \setminus \mathrm{vars}(\vec{x})$. $\in_\alpha$ indicates that some alphabetic variant of the constraint appears in F, and ϕ in this case is taken to be the propositional equivalent of the existentially quantified formula; this removes the effect of the variables that do not appear in the head of the rule.

By definition $U_P \propto T_P$, so naively we would expect that $lfp\ U_P^{\mathcal{E}} \propto lfp\ T_P^E$ whenever $\mathcal{E} \propto E$, and so a finiteness analysis may be performed by computing $lfp\ U_P^{\mathcal{E}}$. However we shall see that $lfp\ U_P^{\mathcal{E}} \not\propto lfp\ T_P^E$ for the program P

```
p(0).
p(X) :- s(X,Y), p(Y).
```

in which s is a extensive predicate intended to be the successor relation. The associated integrity constraint is $\mathcal{E} = \{s(x, y) : x \leftrightarrow y\}$. Now

$$\begin{aligned}
U_P^{\mathcal{E}} \uparrow 0 &= \{p(x) : false, s(x, y) : false\} \\
U_P^{\mathcal{E}} \uparrow 1 &= \{p(x) : false, s(x, y) : x \leftrightarrow y\} \\
U_P^{\mathcal{E}} \uparrow 2 &= \{p(x) : x, s(x, y) : x \leftrightarrow y\} \\
U_P^{\mathcal{E}} \uparrow 3 &= U_P^{\mathcal{E}} \uparrow 2
\end{aligned}$$

So $(p(x) : x) \in lfp\ U_P^{\mathcal{E}}$, which leads to the incorrect conclusion that p is finite for P and $\mathcal{E}$.

The problem arises because:

Proposition 2.4 $FDesc$ is not admissible.

Proof: Let $a_1, a_2, \ldots$ be different constant symbols. Define relation R_β to be $\{p(a_i) \mid i \leq \beta\}$ where β is either finite or the first infinite ordinal ω. Let $\Re$ be $\{p(x) : x\}$ indicating that p is a finite relation. Let C be the chain $\bigcup_{i \leq \omega} \langle R_i, \Re \rangle$. Clearly $\sqcup C = \langle R_\omega, \Re \rangle$. Now, for all $i \leq \omega$, $\Re \propto R_i$, but $\Re \not\propto R_\omega$, so $\propto$ is not admissible. ■

3 Inadmissible Predicates and Abstract Interpretation

We have seen that the traditional abstract interpretation approach for developing dataflow analyses does not apply when the description is inadmissible. In this section we suggest three simple techniques to develop analyses in this case. The techniques are illustrated by using them to develop finiteness analyses.

3.1 Using a Stronger Admissible Property

The first approach is based on a simple idea: rather than finding an analysis to directly show that an inadmissible property P' holds, develop an analysis for an admissible property P which implies P'. The approach is strengthened by the observation that we do not require admissibility for all chains in the cpo, only those that will be encountered when computing the least fixpoint. That is, we need only show that the description is admissible for chains based on the Kleene sequence of the operator to be approximated.

We first clarify how we can use the result of one analysis on some description domain to tell us something about the program in another description domain.

Definition. Let D, D' be descriptions for E. Function $\delta : D \rightarrow D'$ is *approximation preserving* if

$$\forall d \in D \,.\, \forall e \in E \,.\, d \propto_D e \Rightarrow (\delta\ d) \propto_{D'} e. \quad ■$$

Given an approximation preserving mapping from description domain D to D' we can use it to map the results of an analysis on D to the descriptions in D'. If D' is singular, then there is a best (in the sense of most precise) approximation preserving function. We call this the "induced approximation preserving function".

Definition. Let $\langle D, \propto, E \rangle$ be a description and $\langle D', \propto', E \rangle$ a singular description with inducement function $\widehat{\alpha}'$. The *induced approximation preserving function* $\delta : D \rightarrow D'$ is defined by $\delta\ d = \sqcup\{\widehat{\alpha}'\ e \mid d \propto e\}$. ■

The following definition makes precise what it means for a description to be admissible for a particular operator.

Definition. The description $\langle D, \propto, E \rangle$ is *admissible* for $F : E \rightarrow E$ if $\propto$ is admissible on $D \times E'$ where $E' \subseteq E$ is the Kleene sequence of F. ■

Correctness of this technique is captured by the following theorem:

Theorem 3.1 Let D, D' be descriptions for E and $\delta : D \rightarrow D'$ be approximation preserving. Let $F_D : D \rightarrow D$ approximate $F_E : E \rightarrow E$. If D is admissible for F_E, then $\delta\ (lfp\ F_D) \propto_{D'} lfp\ F_E$.

Proof: It follows from Proposition 2.2 that $lfp\ F_D \propto_D lfp\ F_E$. Thus the result follows from the definition of approximation preserving. ■

One might hope that for a given inadmissible property P there is a "best" or "strongest" admissible property implying P. Unfortunately this is not necessarily true as the property of being admissible is itself an inadmissible property.

We now give two finiteness analyses which illustrate this technique. The first example relies on the simple observation that for operators with a finite Kleene sequence, all descriptions are admissible.

FINITENESS ANALYSIS 1: WEAKLY-BOUNDED PROGRAMS

If we can guarantee that the program P being analyzed is *weakly-bounded* in the sense that for all E satisfying the integrity constraints, T_P^E has a finite Kleene sequence, then any description is admissible for T_P^E, and in particular $FDesc$ is. Thus we can analyze whether Q is finite for P by computing $lfp\ U_P^{\mathcal{E}}$ and checking that this implies finiteness. More formally:

Theorem 3.2 Let P be weakly-bounded for the integrity constraint $\mathcal{E}$. Then predicate Q is finite for P and $\mathcal{E}$ if $(Q(\vec{x}) : \bigwedge \vec{x}) \in lfp\ U_P^{\mathcal{E}}$. ■

This idea is essentially the same as that behind finiteness analyses based on "weak finiteness", which was introduced by Sagiv and Vardi [18]. The hard part in this type of analysis is showing that a program is weakly-bounded. Many analyses to detect sub-cases of weak-boundedness are found in the database literature. There are various notions of boundedness (e.g., see [6, 7, 8, 14, 15]), and Naughton and Sagiv give a decision procedure for a class of bounded recursions [15]. Clearly boundedness implies weak boundedness.

FINITENESS ANALYSIS 2: CONTAINMENT

The second example is more complex. The underlying idea is that given a fixed finite set S say, the property of being contained in S is admissible, and clearly containment in a finite set implies finiteness. Descriptions of containment can be simply captured using propositional variables and formulas much as we used these to capture finiteness. Thus the description $p(x, y) : x \vee y$ is read as "the values in the first argument of the relation assigned to p are contained in S or the values in the second argument of the relation are contained in S". However the key problem is how to choose the set S so that it gives good results for all programs and EDBs. Our solution is to make the choice of S dependent on the current relation assigned to the extensional predicates. We choose S to be the set of all arguments which come from an attribute in the EDB which has a finite number of values. More formally,

Definition. Relation R for predicate Q *C-satisfies* predicate description $Q(\vec{x}) : \phi$ for set B iff

$$\forall R' \subseteq R \,.\, R' \neq \emptyset \Rightarrow \tau_{R'}\ satisfies\ \phi$$

where the truth assignment $\tau_{R'}$ to the variables $\vec{x}$ is given by $(\tau_{R'}\ \vec{x}_i)$ iff $R'_i \subseteq B$. Relation R *C-satisfies* relation description $\Re$, written $R \models_c \Re$ (or $\Re \dashv_c R$), iff for all predicates Q, R_Q satisfies $\Re_Q$ for B_R where

$$B_R = \cup\{R_{Q,i} \mid Q \in EP \text{ and } R_{Q,i} \text{ is finite}\}.$$

The *finite containment description* $CDesc$ is the description $\langle RDesc, \dashv_c, \wp HB\rangle$.

■

Lemma 3.3 $CDesc$ is singular. ■

The descriptions $CDesc$ are admissible for any T_P^E as the choice of the bounding set R_B essentially does not change in the Kleene sequence. Of course making the choice of R_B dependent on the current relations assigned to the intensional predicates leads to an inadmissible description for T_P^E.

Lemma 3.4 $CDesc$ is admissible for T_P^E. ■

Definition. Let P be a program. Define $V_P : RDesc \rightarrow RDesc \rightarrow RDesc$ to be the function induced from $T_P : \wp\, HB \rightarrow \wp\, HB \rightarrow \wp\, HB$ using the description $CDesc$. ■

As $CDesc$ is admissible for T_P^E, we can analyze whether Q is finite for P by computing $lfp\ V_P^{\mathcal{E}}$ and checking that this implies finiteness. More formally:

Definition. $\delta_{FC} : RDesc \rightarrow RDesc$ is the induced approximation preserving function from $FDesc$ to $CDesc$. ■

Theorem 3.5 Predicate Q is finite for program P and integrity constraints $\mathcal{E}$ if

$$(Q(\vec{x}) :\wedge \vec{x}) \in lfp\ V_P^{(\delta_{FC}\ \mathcal{E})}. \quad ■$$

While this definition specifies V_P non-constructively in terms of T_P, it is not difficult to see that it gives rise to a terminating dataflow analysis, since V_P is defined on the finite domain $RDesc$.

Example 3.1 Consider the following program taken from Example 7 in [9]:

```
p(X1,Y1) :- d(X1), f(Y1)
p(X2,Y2) :- f(X2), d(Y2)
p(X3,Y3) :- p(X3,Y3)
q(X4) :- p(X4,X4)
```

Let $\mathcal{E}$ be the integrity constraint $\{d(x) : x\}$ stating that EDB d is finite. Now $\delta_{FC}\ \mathcal{E} = \mathcal{E}$. From this, the reader may verify that $(q(x) : x) \in lfp\ V_P^{(\delta_{FC}\ \mathcal{E})}$. Thus, the analysis concludes that q is finite. Note that this example is not handled by the methods in [9] nor any other finiteness analysis that we know of.

We conjecture that the containment analysis when restricted so that only conjunctions of propositional variables are allowed in the descriptions is equivalent to the graph based analysis given in Section 9 of [9].

3.2 Approximate From Above

The second technique is also quite simple. Rather than computing an approximation to the least fixpoint we instead compute an approximation to the greatest fixpoint. Soundness of this approach does not require that the approximation be admissible, since we are already starting from above the least fixpoint. Formal correctness of the technique is captured as a corollary of the following somewhat stronger result.

Theorem 3.6 Let D be a (possibly inadmissible) description of E, and let $F_D : D \rightarrow D$ approximate $F_E : E \rightarrow E$. Let e be any fixpoint of F_E and let $d \propto e$, then: (i) For any finite $k \geq 0$, $F_D{\downarrow}_d^k \propto e$; and (ii) If D is co-admissible, then for any ordinal β, $F_D{\downarrow}_d^\beta \propto e$.

Proof: (Sketch) By hypothesis, $d \propto e$, so since F_D approximates F_E we have $F_D\ d \propto F_E\ e = e$. Using finite induction (1) holds. Co-admissibility of $\propto$ allows us to use transfinite induction to prove (2). ■

Corollary 3.7 Let D be a co-admissible description of E, and let $F_D : D \to D$ approximate $F_E : E \to E$. Then $gfp\ F_D \propto e$ where e is any fixpoint of F_E.

Interesting cases of this corollary are when e is the least fixpoint or when it is the greatest fixpoint. A related technique was suggested by Codish et al. [1] for analyses in which the Kleene sequence for the approximating operator is too long. However, they did not consider inadmissible descriptions.

FINITENESS ANALYSIS 3: APPROXIMATION OF THE GFP

Thus we can analyze whether Q is finite for P by computing $gfp\ U_P^{\mathcal{E}}$ and checking that this implies finiteness.

Lemma 3.8 $FDesc$ is co-admissible. ■

Theorem 3.9 Predicate Q is finite for program P and integrity constraint $\mathcal{E}$ if $(Q(\vec{x}) : \bigwedge \vec{x}) \in gfp\ U_P^{\mathcal{E}}$. ■

Example 3.2 Consider the following program P taken from [9]

```
p(X1,X1) :- f(X1).
p(X2,Y2) :- f(Y2), g(X2,V2), h(X2,W2), p(V2,W2).
p(X3,Y3) :- f(X3), g(Y3,V3), h(Y3,W3), p(V3,W3).
```

with integrity constraints $\mathcal{E} = \{f(x) : x, h(x,y) : y \to x, g(x,y) : y \to x\}$ stating that f represents a finite set, and both h and g are finite in the first argument for subrelations with a finite set of second arguments. Now

$$\begin{array}{rcl}
U_P^{\mathcal{E}} \downarrow 0 & = & \{f(x) : true, g(x,y) : true, h(x,y) : true, p(x,y) : true\} \\
U_P^{\mathcal{E}} \downarrow 1 & = & \mathcal{E} \cup \{p(x,y) : true\} \\
U_P^{\mathcal{E}} \downarrow 2 & = & \mathcal{E} \cup \{p(x,y) : x \vee y\} \\
U_P^{\mathcal{E}} \downarrow 3 & = & \mathcal{E} \cup \{p(x,y) : x \wedge y\} \\
U_P^{\mathcal{E}} \downarrow 4 & = & U_P^{\mathcal{E}} \downarrow 3
\end{array}$$

Thus $(p(x,y) : x \wedge y) \in gfp\ U_P^{\mathcal{E}}$, and the analysis determines that p is finite.

It is interesting to note that the $U_P^{\mathcal{E}}$ operator, when used in approximation from above, is equivalent to the axiomatic superfiniteness analysis given by Kifer et al. in [9]. The authors present four sets of axioms which form a sound and complete system for finding finiteness constraints that hold in all fixpoints of the program:

PC-rules Each PC (partial constraint) corresponds to a disjunction of propositions encoding finiteness information for the described decomposition of a relation. The PC rules follow directly from the axioms of propositional logic.

PRD-rules Projection dependencies correspond to a restriction of finiteness constraints to certain columns in a relation's tuples. This is exactly what is done by the existential quantification of $U_P^{\mathcal{E}}$.

IND-rules An inclusion dependency indicates that one relation is a subrelation of another. The IND rule, which associates with the subrelation the constraints of the containing relation, is captured by $U_P^{\mathcal{E}}$ when the IDB rule which specifies the subrelationship is evaluated.

DD-rules The decomposition dependencies simply gather the dependencies of elements of a decomposition of a relation. This gathering is done in $U_P^{\mathcal{E}}$ by forming the disjunction of the FCs derived by considering the rules for a relation separately.

Now finiteness in all fixpoints holds iff finiteness holds in the greatest fixpoint. Further, using U_P^E starting with the top of the description domain corresponds to making inferences based only on tautologies, and thus is equivalent to the method presented in [9].

3.3 Combination

Our final technique is to combine the two previous techniques. First an admissible description is used to find an approximation to the least fixpoint, then we use this approximation as the start value in the second method and iterate downwards, while still staying above the least fixpoint. From Theorems Theorem 3.1 and Theorem 3.6, we have:

Theorem 3.10 Let D, D' be descriptions for E and $\delta : D \to D'$ be approximation preserving. Let $F_D : D \to D$ and $F_{D'} : D' \to D'$ approximate $F_E : E \to E$. Let D be admissible for F_E and $d = (\delta\ (lfp\ F_D))$. Then: (i) for all finite ordinals k, $F_{D'}{\downarrow}_d^k \propto_{D'} lfp\ F_E$; and (ii) if D is co-admissible, for all ordinals β, $F_{D'}{\downarrow}_d^\beta \propto_{D'} lfp\ F_E$. ∎

The combination approach given here is somewhat related to the widening/ narrowing approach to abstract interpretation introduced by Cousot and Cousot [2]. The widening/narrowing approach was developed to handle the case when the description domain has infinite ascending chains, and so it is not possible to compute the least fixpoint of the approximating operator using a finite Kleene sequence. An extreme use of this technique is when the description domain is just the original domain. The idea is to use a widening operator to jump above the least fixpoint and then a narrowing operator to move downwards from this point.

It is implicit in the widening/narrowing approach of Cousot and Cousot that there is an Galois connection between the original domain and the domain on which the widening and narrowing operator are defined. This ensures that the approximation relation is admissible. Thus our combination approach is, in some senses, a relaxation of widening/narrowing in which the widening and narrowing operators are allowed to work on different domains and the approximation relation for the narrowing operator's domain is not required to be admissible. It is also a specialization, as when couched in these terms we are using the trivial widening operator "lub" and the trivial narrowing operator "glb". However it should be emphasized that the two techniques were developed to handle very different problems.

FINITENESS ANALYSIS 4: COMBINATION
Thus we can analyze whether Q is finite for P by combining Finiteness Analysis 2 with Finiteness Analysis 3.

Definition. $\delta_{CF} : RDesc \to RDesc$ is the induced approximation preserving function from $CDesc$ to $FDesc$. ∎

Theorem 3.11 Predicate Q is finite for program P and integrity constraints $\mathcal{E}$ if $(Q(\vec{x}) : \bigwedge \vec{x}) \in gfp\ W_P^{\mathcal{E}}$, where $W_P^{\mathcal{E}}$ is defined by $W_P^{\mathcal{E}}\ \Re = (\delta_{CF}\ \Re_V) \sqcap U_P^{\mathcal{E}}\ \Re$ and $\Re_V = lfp\ V_P^{(\delta_{FC}\ \mathcal{E})}$. ■

Note that *glb* on *RDesc* is essentially component-wise conjunction of the propositional formula. This analysis, to our knowledge, is the most precise finiteness analysis known.

4 Conclusions

We have systematically investigated the development of dataflow analyses for inadmissible program properties. This is important as some practical program properties, such as finiteness, are inadmissible, and blindly applying traditional techniques from abstract interpretation leads to incorrect analyses. We have illustrated these techniques by means of four finiteness analyses, the last of which is the most precise finiteness analysis that we are aware of.

The advantage of developing finiteness analyses in the framework given here is that it simplifies their description and proof of correctness. Essentially a dataflow analysis is given by simply specifying the description domain and choosing one of the techniques given here. Another advantage is that the analyses can be lifted to the case of non-constant function symbols.

Acknowledgements Discussions with Raghu Ramakrishnan were very helpful in clarifying the relationships between the various finiteness analyses proposed in the deductive database literature.

References

[1] M. Codish, J. Gallagher and E. Shapiro, "Using Safe Approximations of Fixed Points for Analysis of Logic Programs", *Proc. META88, Workshop on Metaprogramming in Logic Programming*, Bristol, June 1988.

[2] P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints", *Proc. Fourth ACM Symp. on Principles of Programming Languages*, 1977.

[3] P. Cousot and R. Cousot, "Systematic Design of Program Analysis Frameworks", *Proc. Sixth ACM Symp. on Principles of Programming Languages*, 1979.

[4] P. Cousot and R. Cousot, "Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation", Manuscript, 1991.

[5] S. K. Debray, "Static Inference of Modes and Data Dependencies in Logic Programs", *ACM Transactions on Programming Languages and Systems* vol. 11, no. 3, June 1989, pp. 419-450.

[6] H. Gaifman, H. Mairson, Y. Sagiv, and M. Vardi, "Undecidable Optimization Problems for Database Logic Programs", *Proc. Second IEEE Symposium on Logic in Computer Science*, Ithaca, 1987.

[7] Y. Ioannidis, "Bounded Recursion in Deductive Databases", Technical Report, UCB/ERL M85.6, University of California, Berkeley, Feb. 1985.

[8] P. Kanellakis, "Logic Programming and Parallel Complexity", Technical Report CS-86-23, Dept. of Computer Science, Brown University, Oct. 1986.

[9] M. Kifer, R. Ramakrishnan, and A. Silberschatz, "An Axiomatic Approach to Deciding Finiteness of Queries in Deductive Databases", manuscript (Preliminary version appeared in *Proc. Seventh ACM Symp. on Principles of Deductive Databases*, Austin, TX, March 1988).

[10] K. Marriott. Frameworks for abstract interpretation. To appear *Acta Informatica.*

[11] K. Marriott and H. Søndergaard. Bottom-up abstract interpretation of logic programs. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proc. Fifth Int. Conf. Symp.*, pages 733–748. MIT Press, 1988.

[12] K. Marriott and H. Søndergaard. Bottom-up dataflow analysis of normal logic programs. To appear *The Journal of Logic Programming.*

[13] C. S. Mellish, "Abstract Interpretation of Prolog Programs", *Proc. Third International Conference on Logic Programming*, London, July 1986. Springer-Verlag LNCS vol. 225.

[14] J. Naughton, "Data Independent Recursion in Deductive Databases", *Proc. Fifth ACM Symp. on Principles of Database Systems*, March 1986.

[15] J. Naughton and Y. Sagiv, "A Decidable Class of Bounded Recursions", *Proc. Sixth ACM Symp. on Principles of Database Systems*, San Diego, CA, March 1987.

[16] F. Nielson. Strictness analysis and denotational abstract interpretation. *Information and Computation* **76** (1) : 29–92, 1988.

[17] R. Ramakrishnan, F. Bancilhon, and A. Silberschatz, "Safety of Recursive Horn Clauses with Infinite Relations", *Proc. ACM Symp. on Principles of Database Systems*, 1987.

[18] Y. Sagiv and M. Y. Vardi, "Safety of Datalog Queries over Infinite Databases", *Proc. Eighth ACM Symposium on Principles of Database Systems*, Philadelphia, PA, March 1989.

[19] H. Søndergaard. An application of abstract interpretation of logic programs: Occur check reduction. In B. Robinet and R. Wilhelm, editors, *Proc. ESOP 86* (Lecture Notes in Computer Science 213), pages 327–338. Springer-Verlag, 1986.