Information Processing Letters 47 (1993) 149–157
Elsevier

14 September 1993

Efficient bottom–up execution of logic programs using abstract interpretation

Byeong-Mo Chang, Kwang-Moo Choe and Taisook Han

Department of Computer Science, Korea Advanced Institute of Science and Technology, 373-1, Kusung-dong, Yusung-ku, Taejon 305-701, South Korea

Communicated by K. Ikeda
Received 23 April 1993
Revised 13 May 1993

Keywords: Applicative programming; filtering; system graph; abstract filter; abstract interpretation

1. Introduction

Bottom–up evaluation of logic programs has recently attracted much attention in the logic programming and the deductive database field. It is complete [7], and if the number of all possible facts is finite, it is guaranteed to terminate, while top–down evaluation may not. Bottom–up evaluation, however, may be inefficient since it may generate many facts irrelevant to the query. To overcome this drawback, a number of optimization strategies have been proposed on seminaive bottom–up evaluation [1,6,7,9].

Filtering is a query optimization strategy based on the seminaive evaluation on *system graphs*, which evaluates a query in terms of the bottom–up data flow on system graphs. Filtering methods try to restrict data flow on system graphs as much as possible by means of filters, which are selections attached on arcs of system graphs [3,6,7]. *Static filtering* restricts data flow using *static filters* which are computed at compile-time by propagating data-independent bindings. *Dynamic filtering* uses dynamic filters which are computed by propagating "actual data" generated during evaluation [6].

In this paper, we propose a new filter called *abstract filter* which is computed at compile time using the result of a *two-phase abstract interpretation*. The two-phase abstract interpretation consists of a bottom–up phase followed by a top–down one, and it provides an approximation of (possibly nonground) success patterns *relevant* to a given query. The bottom–up phase extends the bottom–up abstract interpretation in [2] following the framework in [8]. A two-phase abstract interpretation is not a new idea in this area even though it has not been published in our form. The abstract filter prevents facts not in that approximation from being passed in the evaluation of system graphs. From a theoretical point of view, the abstract filter is shown to be at least as powerful as the static filter under some assumption.

The next section gives preliminaries. Section 3 describe filtering on system graphs. Section 4 describes abstract interpretation. In Section 5, the abstract filter is computed. Section 6 concludes the paper.

Correspondence to: B.-M. Chang, Department of Computer Science, Korea Advanced Institute of Science and Technology, 373-1, Kusung-dong, Yusung-ku, Taejon 305-701, South Korea. Email: chang@plhae.konist.ac.kr.

2. Preliminaries

Let *Pred*, *Cons* and *Var* denote the set of predicates, the set of constructors (constants or function symbols) and a denumerable set of variables, respectively. The set of all terms constructed from *Cons* and *Var* is denoted by *Term*(*Cons*, *Var*). *Atoms* is the set of all atoms $p(t_1,\ldots,t_n)$ constructed from $p \in Pred$ and $t_1,\ldots,t_n \in Term(Cons, Var)$. A *program* P is a finite set of *program clauses* of the form $A \leftarrow B_1,\ldots,B_n$ $(n \geqslant 0)$ where A is a head atom and $B_1,\ldots,B_n$ are body atoms. If $n = 0$ then the clause is simply $A \leftarrow$ and called a *fact*. Otherwise it is called a *rule*. A rule is denoted by δ if necessary. A function $pred(\delta, i)$ is defined to be the ith predicate symbol in the body of δ, and a function $head(\delta)$ is the predicate of the head of δ. A *query* Q is of the form $\leftarrow B_1,\ldots,B_n$ where $B_1,\ldots,B_n$ are atoms.

A partial function *mgu* maps a pair of syntactic objects to an indempotent most general unifier of them. We write $mgu(\langle a_1,\ldots,a_n\rangle, \langle b_1,\ldots,b_n\rangle)$ to denote the most general unifier of the set of equations $\{a_1 = b_1,\ldots,a_n = b_n\}$. When S is a set and $\sim$ is an equivalence relation on S, $S/\sim$ is the set of equivalence classes on S with respect to $\sim$. For an element $a \in S$, $[a]_\sim$ denotes the equivalence class of a with respect to $\sim$, that is, $[a]_\sim = \{a' \mid a' \sim a\}$.

The *extended Herbrand universe* U_P is $Term(Cons, Var)/\sim$ where $\sim$ is the *variance* relation (i.e. $t_1 \sim t_2$ iff $\exists \vartheta_1,\vartheta_2\ t_1\vartheta_1 = t_2 \wedge t_2\vartheta_2 = t_1$). The variance relation can be extended easily to any syntactic object (atoms, clauses etc.). The *extended Herbrand base* B_P *is* $Atoms/\sim$ where $\sim$ is the variance relation extended on atoms. An interpretation I is a subset of B_P. Let $\wp(S)$ denote the power set of a set S. Given an equivalence class $[A]_\sim$ of atoms and a finite set of variables V, it is always possible to find a representative A' in $[A]_\sim$ that contains no variables from V. For a syntactic object s and a set of equivalence classes of atoms I, we denote by $\langle A_1,\ldots,A_n\rangle \ll_s I$ that $A_1,\ldots,A_n$ are representatives of elements of I renamed apart from s and from each other. For simplicity, an atom represents its equivalence class and we write A rather than $[A]_\sim$.

Definition 1 [5]. $T_P : \wp(B_P) \to \wp(B_P)$ *is a transformation associated with a program* P,

$$Tp(I) = \left\{ A\vartheta \;\middle|\; \begin{array}{l} C = A \leftarrow B_1,\ldots,B_n \in P \\ \langle B_1',\ldots,B_n'\rangle \ll_C I \\ \vartheta = mgu(\langle B_1,\ldots,B_n\rangle, \langle B_1',\ldots,B_n'\rangle) \end{array} \right\}.$$

The semantics of a program P is determined by $lfp(T_P) = T_P^\omega$, the least fixpoint of T_P [5].

The abstract interpretation framework [4] presupposes a least fixpoint characterization of the collecting semantics. We assume a concrete interpretation can be defined in terms of a monotonic operator $E_P : E \to E$ on a domain E, and an abstract analysis in terms of a monotonic operator $D_P : D \to D$ on an abstract domain D. We assume the elements of E and D are sets.

Definition 2 [4,8]. An *abstract interpretation* $((E, \subseteq), E_P, (D, \subseteq), D_P, \alpha, \gamma)$ consists of a complete lattice $(E, \subseteq)$, a monotonic operator $E_P : E \to E$, a complete lattice $(D, \subseteq)$, a monotonic operator $D_P : D \to D$, an abstraction function $\alpha : E \to D$ and a concretization function $\gamma : D \to E$, such that

(1) α and γ are monotonic,
(2) $d = \alpha(\gamma(d))$ for all $d \in D$,
(3) $e \subseteq \gamma(\alpha(e))$ for all $e \in E$, and
(4) $E_P(\gamma(d)) \subseteq \gamma(D_P(d))$ for all $d \in D$.

Conditions (1)–(3) state that $((E, \subseteq), \alpha, (D, \subseteq), \gamma)$ forms a *Galois insertion*. Condition (4) is the *safeness* criterion that ensures that D_P faithfully mimics E_P.

Proposition 3 [4]. *If* $((E, \subseteq), Ep, (D, \subseteq), Dp, \alpha, \gamma)$ *is an abstract interpretation,* $\gamma(lfp(D_P)) \subseteq lfp(E_P)$.

3. Filtering on system graphs

A *system graph* for a program P was introduced as a tool for bottom-up query evaluation [6,7]. We assume that a query Q is represented as $ans \leftarrow Q$ and it is contained in P. Let $Arg\_Var(\delta, i)$ denote the *argument variable* of the ith body atom of a rule δ. An argument variable is a variable corresponding to an argument position of an atom. If an atom has an m-ary predicate p, its argument variables are $P1, \ldots, Pm$ named after the predicate symbol. We denote by n_δ the number of body atoms in a rule δ.

Definition 4. A *system graph for a program* P is $Sg = (V_P, V_R, E_{P,R}, E_{R,P}, F)$ where
V_P and V_R are the sets of predicates and rules in P respectively,
$E_{P,R} = \{(p, \delta)/i \mid p \in V_P, \delta \in V_R, p = pred(\delta, i), 1 \leqslant i \leqslant n_\delta\}$,
$E_{R,P} = \{(\delta, p) \mid \delta \in V_R, p \in V_P, p = head(\delta)\}$,
F is a *filter function* that associates a filter over $Arg\_Var(\delta, i)$ with every arc $(p, \delta)/i \in E_{P,R}$.

The slashed pair $(p, \delta)/i$ in $E_{P,R}$ denotes the ith input port (incoming arc) of a rule-node δ from a pred-node p. The value of a filter function F on a port $(p, \delta)/i$, $F((p, \delta)/i)$, is called the *filter* of that port and it represents a logical formula over $Arg\_Var(\delta, i)$ which tuples must satisfy to pass the port. A filter $F((p, \delta)/i)$ is denoted by $F_{\delta,i}$ in short. The system graph with a filter function F for a program P is denoted by SG_P^F. Figure 1 shows the system graph for the program in Example 7.

The simplistic bottom-up evaluation evaluates a query in terms of bottom-up data flow on system graphs [7]. The filter of each port in $E_{P,R}$ restricts data flow through the port in the evaluation by not passing facts or its tuples if they do not satisfy the filter. The evaluation on SG_P^F is defined formally in terms of a new operator T_P^F. An atom $B = p(t_1, \ldots, t_m)$ is denoted by $B \in F_{C,i}$ if it satisfies a filter $F_{C,i}$.

Definition 5. $T_P^F : \wp(B_P) \to \wp(B_P)$,

$$T_P^F(I) = \left\{ A\vartheta \;\middle|\; \begin{array}{l} C = A \leftarrow B_1, \ldots, B_n \in P \\ \langle B_1', \ldots, B_n' \rangle \ll_C I \\ B_i' \in F_{C,i} (1 \leqslant i \leqslant n),\ \vartheta = mgu(\langle B_1, \ldots, B_n \rangle, \langle B_1', \ldots, B_n' \rangle) \end{array} \right\}.$$

The evaluation on SG_P^F computes $lfp(T_P^F)$, that is, each pred-node contains its fact in $lfp(T_P^F)$.

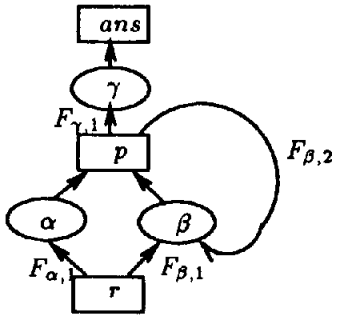

Fig. 1. System graph for the program in Example 7.

We review the static filter computation in [7] in brevity.

Definition 6 [10]. A *level-k subterm* is defined as follows:
(a) For a given term t, t is a level-0 subterm of t.
(b) If a level-k subterm of t is $f(t_1,\ldots,t_n)$, then t_i is a level $k+1$ subterm of t.

The *depth-k abstract term* of a term t is the term t' which is obtained by substituting every level-k subterm of t with a *fresh variable* [10]. *The k-signature* of a term extends the depth-k abstraction by maintaining symbols in the deleted subterms as a set [7]. It propagates bindings in the query backward on system graph until no more new calling (binding) patterns are found, with every calling pattern being maintained in its k-signature form. We denote the static filter by *SF* in the following. See [7] for details.

Example 7. Consider the static filter computation for the following program P:

$$\gamma: \leftarrow p(X, f(f(b))). \quad (\textit{query})$$
$$\alpha: p(X, Y) \leftarrow r(X, Y).$$
$$\beta: p(X, f(Y)) \leftarrow r(X, Z),\ p(Z, Y).$$

The selection $P2 = f(f(b))$ in the query imposes a selection $P2 = f(b)$ on $(p, \beta)/2$ if pushed backward via the rule β. The new selection $P2 = f(b)$ imposes a selection $P2 = b$ if pushed backward via β. The selections $P2 = f(f(b))$, $P2 = f(b)$ and $P2 = b$ can be pushed similarly via the rule α. The static filters computed are $SF_{\beta,2} = (P2 = f(b) \vee P2 = b)$. and $SF_{\alpha,1} = (R2 = f(f(b)) \vee R2 = f(b) \vee R2 = b)$.

4. Abstract interpretation

We first review a basic bottom-up abstract interpretation $((\wp(B_P), \subseteq), T_P, (\wp(B_P^{\mathscr{A}}), \subseteq), T_P^{\mathscr{A}}, \alpha_a, \gamma_a)$ in [2]. Let $\rho(t)$ be a function which maps a term t to its depth-k abstract term. Then *abstract universe* of a program P is defined by $U_P^{\mathscr{A}} = \rho(U_P)$. *Abstract base* $B_P^{\mathscr{A}}$ of a program P is defined by $Atoms^{\mathscr{A}}/\sim$ where $Atoms^{\mathscr{A}} = \{p(t_1^{\mathscr{A}},\ldots,t_n^{\mathscr{A}}) \mid p \in Pred^n, \{t_1^A,\ldots,t_n^A\} \subseteq U_P^{\mathscr{A}}\}$. The abstraction function $\alpha_a: B_P \to B_P^{\mathscr{A}}$ is defined by $\alpha_a(p(t_1,\ldots,t_n)) = [p(\rho(t_1),\ldots,\rho(t_n))]_\sim$, and can be extended to $\alpha_a: \wp(B_P) \to \wp(B_P^{\mathscr{A}})$ which is defined by $\alpha_a(I) = \{\alpha_a(A) \mid A \in I\}$. The corresponding concretization function $\gamma_a: \wp(B_P^{\mathscr{A}}) \to \wp(B_P)$ is $\gamma_a(I^{\mathscr{A}}) = \{A' \in B_P \mid A^{\mathscr{A}} \in I^{\mathscr{A}},\ \alpha_a(A') = A^{\mathscr{A}}\}$.

Lemma 8. $((\wp(B_P), \subseteq), \alpha_a, (\wp(B_P^{\mathscr{A}}), \subseteq), \gamma_a)$ *is a Galois insertion.*

An *abstract substitution* is a substitution $\vartheta^A: V \to U_P^{\mathscr{A}}$ where V is a set of variables. Given a substitution $\vartheta = \{t_1/x_1,\ldots,t_n/x_n\}$, α_Φ is defined by $\alpha_\Phi(\vartheta) = \vartheta^{\mathscr{A}} = \{\rho(t_1)/x_1,\ldots,\rho(t_n)/x_n\}$. The abstract operator $T_P^{\mathscr{A}}$ approximates T_P.

Definition 9 [2]. $T_P^{\mathscr{A}}: \wp(B_P^{\mathscr{A}}) \to \wp(B_P^{\mathscr{A}})$,

$$T_P^{\mathscr{A}}(I^{\mathscr{A}}) = \left\{ \alpha_a(A\vartheta^{\mathscr{A}}) \;\middle|\; \begin{array}{l} c = A \leftarrow B_1,\ldots,B_n \in P, \\ \langle B_1^{\mathscr{A}},\ldots,B_n^{\mathscr{A}}\rangle \ll_c I^{\mathscr{A}}, \\ \vartheta^{\mathscr{A}} = mgu^{\mathscr{A}}(\langle B_1,\ldots,B_n\rangle, \langle B_1^{\mathscr{A}},\ldots,B_n^{\mathscr{A}}\rangle) \end{array} \right\},$$

where $mgu^{\mathscr{A}}(\langle B_1,\ldots,B_n\rangle, \langle B_1^{\mathscr{A}},\ldots,B_n^{\mathscr{A}}\rangle) = \alpha_\Phi(\vartheta)$, $\vartheta = mgu(\langle B_1,\ldots,B_n\rangle, \langle B_1^{\mathscr{A}},\ldots,B_n^{\mathscr{A}}\rangle)$.

It is shown in [1] that the least fixpoint $lfp(T_P^{\mathscr{A}})$ is $(T_P^{\mathscr{A}})^n(\emptyset)$ for some finite n, and the analysis is correct.

Theorem 10 [2]. $\gamma(lfp(T_P^{\mathscr{A}})) \supseteq lfp(T_P)$ *for any program* P.

We now describe a two-phase abstract interpretation which consists of a bottom–up phase followed by a top–down one. Let C_P be the set of all instances of clauses in a program P. The set of all *abstract clauses* of P is defined by $C_P^{\mathscr{A}} = \{\alpha_c(C\vartheta^{\mathscr{A}}) \mid C \in P,\ \vartheta^{\mathscr{A}} : Var(C) \to U_P^{\mathscr{A}}\}$ where $Var(C)$ is the set of variables in C and $\alpha_c(A \leftarrow B_1, \ldots, B_n)$ is $[\alpha_a'(A) \leftarrow \alpha_a'(B_1), \ldots, \alpha_a'(B_n)]_\sim$ where $\alpha_a'(B_i)$ maps B_i into its depth-k abstract atom. We first consider the bottom–up phase. It is defined by the abstract interpretation $((\wp(B_P) \times \wp(C_P),\ \subseteq),\ \mathscr{U}_P,\ (\wp(B_P^{\mathscr{A}}) \times \wp(C_P^{\mathscr{A}}),\ \subseteq),\ \mathscr{U}_P^{\mathscr{A}},\ \alpha,\ \gamma)$ which extends the above basic analysis based on the framework in [8]. The concrete operator $\mathscr{U}_P$ is defined by extending T_P.

Definition 11. $\mathscr{U}_P : \wp(B_P) \times \wp(C_P) \to \wp(B_P) \times \wp(C_P)$, $\mathscr{U}_P(I, J) = (T_P(I), J')$ *where*

$$J' = \left\{ C\vartheta \;\middle|\; \begin{array}{l} C = A \leftarrow B_1, \ldots, B_n \in P \\ \langle B_1', \ldots, B_n' \rangle \ll_C I \\ \vartheta = mgu(\langle B_1, \ldots, B_n \rangle, \langle B_1', \ldots, B_n' \rangle) \end{array} \right\}.$$

The concrete collecting semantics of a program P is $lfp(\mathscr{U}_P)$.

The abstraction function $\alpha : \wp(B_P) \times \wp(C_P) \to \wp(B_P^{\mathscr{A}}) \times \wp(C_P^A)$ is defined by $\alpha(I, J) = (\alpha_a(I), \alpha_c(J))$. The concretization function $\gamma : \wp(B_P^{\mathscr{A}}) \times \wp(C_P^{\mathscr{A}}) \to \wp(B_P) \times \wp(C_P)$ is $\gamma(I^{\mathscr{A}}, J^{\mathscr{A}}) = (\gamma_a(I^{\mathscr{A}}), \gamma_c(J^{\mathscr{A}}))$ where $\gamma_c(J^{\mathscr{A}}) = \{C \mid C^{\mathscr{A}} \in J^{\mathscr{A}},\ \alpha_c(C) = C^{\mathscr{A}}\}$. The abstract operator $\mathscr{U}_P^{\mathscr{A}}$ is defined by extending $T_P^{\mathscr{A}}$, so as to contain abstract clauses as well as abstract atoms.

Definition 12. $\mathscr{U}_P^{\mathscr{A}} : \wp(B_P^{\mathscr{A}}) \times \wp(C_P^{\mathscr{A}}) \to \wp(B_P^{\mathscr{A}}) \times \wp(C_P^{\mathscr{A}})$, $\mathscr{U}_P^{\mathscr{A}}(I^{\mathscr{A}}, J^{\mathscr{A}}) = (T_P^{\mathscr{A}}(I^{\mathscr{A}}), J^{\mathscr{A}'})$ *where*

$$J^{\mathscr{A}'} = \left\{ \alpha_c(C\vartheta^{\mathscr{A}}) \;\middle|\; \begin{array}{l} C = A \leftarrow B_1, \ldots, B_n \in P, \\ \langle B_1^{\mathscr{A}}, \ldots, B_n^{\mathscr{A}} \rangle \ll_C I^{\mathscr{A}}, \\ \vartheta^{\mathscr{A}} = mgu^{\mathscr{A}}(\langle B_1, \ldots, B_n \rangle, \langle B_1^{\mathscr{A}}, \ldots, B_n^{\mathscr{A}} \rangle) \end{array} \right\}.$$

The bottom–up phase computes the least fixed point $lfp(\mathscr{U}_P^{\mathscr{A}})$ which is shown to terminate in the following. Its first part is denoted by $lfp_a(\mathscr{U}_P^{\mathscr{A}})$ and the second part by $lfp_c(\mathscr{U}_P^{\mathscr{A}})$ is the same as $lfp(T_P^{\mathscr{A}})$ and $lfp_c(\mathscr{U}_P^{\mathscr{A}})$ contains all abstract clauses whose body goals are in $lfp(T_P^{\mathscr{A}})$.

Theorem 13. *Let* P *be a logic program. The least fixpoint* $lfp(\mathscr{U}_P^{\mathscr{A}}) = (\mathscr{U}_P^{\mathscr{A}})^n(\emptyset)$ *for some finite* n.

Proof. Since $\mathscr{U}_P^{\mathscr{A}}$ is monotone, $lfp(\mathscr{U}_P^A)$ exists by the Knaster–Tarksi theorem, and since $\wp(B_P^{\mathscr{A}} \times C_P^{\mathscr{A}})$ is a finite lattice, $lfp(\mathscr{U}_P^{\mathscr{A}}) = (\mathscr{U}_P^{\mathscr{A}})^n(\emptyset)$ for some finite n. □

The following lemma and theorem show the safeness of the bottom–up phase.

Lemma 14. $\gamma(\mathscr{U}_P^{\mathscr{A}}(I^{\mathscr{A}}, J^{\mathscr{A}})) \supseteq \mathscr{U}_P(\gamma(I^{\mathscr{A}}, J^{\mathscr{A}}))$ *for all* $I^{\mathscr{A}} \in \wp(B_P^{\mathscr{A}})$, $J^{\mathscr{A}} \in \wp(C_P^{\mathscr{A}})$.

Proof. See [3] for the proof. □

Theorem 15. $\gamma(lfp(\mathscr{U}_P^{\mathscr{A}})) \supseteq lfp(\mathscr{U}_P)$ *for any program* P.

Proof. It is straightforward that $((\wp(B_P) \times \wp(C_P), \subseteq), \alpha, (\wp(B_P^{\mathscr{A}}) \times \wp(C_P^{\mathscr{A}}), \subseteq), \gamma)$ is a Galois insertion. So the proof is by Lemma 14 and Proposition 3. □

Example 16. Let P be the following logic program:

$$\alpha: path(X, [X]) \leftarrow final(X).$$
$$\beta: path(X, [X \mid P]) \leftarrow arc(X, N), path(N, P).$$
$$final(f).$$
$$arc(a, b).\ arc(a, c).\ arc(c, b).\ arc(c, d).\ arc(d, f).\ arc(e, c).\ arc(e, f).\ arc(g, e).$$

Let the depth of abstraction be 2. The result of the bottom–up phase is as follows.

$$lfp_a(\mathscr{U}_P^{\mathscr{A}}) = \left\{ \begin{array}{l} arc(a,b), arc(a,c), arc(c,b), arc(d,f), arc(e,c), arc(e,f), arc(g,e), final(f). \\ path(f,[f]), path(e,[e\mid -]), path(d,[d\mid -]), path(c,[c\mid -]), path(g,[g\mid -]), path(a,[a\mid -]) \end{array} \right\}$$

$$lfp_c(\mathscr{U}_P^{\mathscr{A}}) = \left\{ \begin{array}{l} arc(a,b)., arc(a,c)., arc(c,b)., arc(c,d)., arc(d,f)., arc(e,c)., arc(e,f)., arc(g,e)., \\ final(f)., \quad path(f,[f]) \leftarrow final(f)., \quad path(e,[e\mid -]) \leftarrow arc(e,f), path(f,[f])., \\ path(d,[d\mid -]) \leftarrow arc(d,f), path(f,[f])., \quad path(c,[c\mid -]) \leftarrow arc(c,d), path(d,[d\mid -])., \\ path(e,[e\mid -]) \leftarrow arc(e,c), path(c,[c\mid -])., \quad path(g,[g\mid -]) \leftarrow arc(g,e), path(e,[e\mid -]). \\ path(a,[a\mid -]) \leftarrow arc(a,c), path(c,[c\mid -]). \end{array} \right\}$$

The facts of path in $lfp_a(\mathscr{U}_P^{\mathscr{A}})$ represent vertices, which are connected to the vertex f. The number of the facts of path in $lfp_a(\mathscr{U}_P^{\mathscr{A}})$ is bounded by the number of vertices, see Section 6 for details.

The bottom–up phase computes approximated success patterns without regard to a query. So, some of them are not relevant to a given query. An abstract clause $H^{\mathscr{A}} \leftarrow B_1^{\mathscr{A}}, \ldots, B_n^{\mathscr{A}} \in lfp(\mathscr{U}_P^{\mathscr{A}})$ is *relevant* to a query Q if

(a) $H^{\mathscr{A}}$ is unifiable with some atoms in Q

(b) $H^{\mathscr{A}}$ is unifiable with some body atom of some abstract clause relevant to Q.

The top–down phase finds a subset of $lfp_c(\mathscr{U}_P^{\mathscr{A}})$, which consists of only the abstract clauses relevant to a query. We will define an abstract interpretation $((\wp(lfp_c(\mathscr{U}_P)), \subseteq), \mathscr{D}_P, (\wp(lfp_c(\mathscr{U}_P^{\mathscr{A}})), \subseteq), \mathscr{D}_P^{\mathscr{A}}, \alpha_c, \gamma_c)$ for the top–down phase. The concrete operator $\mathscr{D}_P$ is defined for a concrete collecting semantics.

Definition 17. $\mathscr{D}_P: \wp(lfp_c(\mathscr{U}_P)) \to \wp(lfp_c(\mathscr{U}_P))$. Consider a given query Q. Let *body-atoms*(J) be the set of body atoms of clauses in J.

$$\mathscr{D}_P(J) = \{C \mid C = A \leftarrow B_1, \ldots, B_n \in lfp_c(\mathscr{U}_P),\ A' \in body\text{-}atoms(Q \cup J),\ \exists mgu(A, A')\}$$

The abstract operator $\mathscr{D}_P^{\mathscr{A}}$ is defined for the top–down phase.

Definition 18. $\mathscr{D}_P^{\mathscr{A}}: \wp(lfp_c(\mathscr{U}_P^{\mathscr{A}})) \to \wp(lfp_c(\mathscr{U}_P^{\mathscr{A}}))$. Consider a given query Q. Let *body-atoms*$(J^{\mathscr{A}})$ be defined as Definition 17.

$$\mathscr{D}_P^{\mathscr{A}}(J^{\mathscr{A}}) = \{C^{\mathscr{A}} \mid C^{\mathscr{A}} = A^{\mathscr{A}} \leftarrow B_1^{\mathscr{A}}, \ldots, B_n^{\mathscr{A}} \in lfp_c(\mathscr{U}_P^{\mathscr{A}}),\ A^{\mathscr{A}'} \in body\text{-}atoms(Q \cup J^{\mathscr{A}}),\ \exists mgu(A^{\mathscr{A}}, A^{\mathscr{A}'})\}$$

The least fixpoint $lfp(\mathscr{D}_P^{\mathscr{A}})$ is $(\mathscr{D}_P^{\mathscr{A}})^n(\emptyset)$ for some finite n because $\mathscr{D}_P^{\mathscr{A}}$ is monotone and $\wp(lfp_c(\mathscr{U}_P^{\mathscr{A}}))$ is a finite lattice. It contains all the abstract clauses in $lfp(\mathscr{U}_P^{\mathscr{A}})$ which are relevant to the query.

Example 19. Consider a query $\leftarrow p(a, 8)$. The top–down phase for the program in Example 16 is

$$lfp(D_P^{\mathscr{A}}) = \left\{ \begin{array}{l} path(a,[a|-]) \leftarrow arc(a,c),\ path(c,[c|-]).,\quad path(c,[c|-]) \leftarrow arc(c,d),\ path(d,[d|-]).,\\ path(d,[f|-]) \leftarrow arc(d,f),\ path(f,[f]).,\quad path(f,[f]) \leftarrow final(f).,\\ final(f).,\ arc(a,c).,\ arc(c,d).,\ arc(d,f). \end{array} \right\}.$$

The facts of path in $lfp(D_P^{\mathscr{A}})$ represent vertices, which are connected to the vertices a and f.

We prove the safeness of the top–down phase in the following lemma and theorem.

Lemma 20. $\gamma_c(\mathscr{D}_P^{\mathscr{A}}(J^{\mathscr{A}})) \subseteq \mathscr{D}_P(\gamma_c(J^{\mathscr{A}}))$ *for all* $J^{\mathscr{A}} \in \wp(G_P^{\mathscr{A}})$.

Proof. See [3] for the proof. □

Theorem 21. $\gamma(lfp(\mathscr{D}_P^{\mathscr{A}})) \subseteq lfp(\mathscr{D}_P)$ *for any program* P.

Proof. By Galois insertion $((\wp(lfp_c(\mathscr{U}_P)), \subseteq), \alpha_c, (\wp(lfp_c(\mathscr{U}_P^{\mathscr{A}})), \subseteq), \gamma_c)$, Lemma 20 and Proposition 3. □

5. Abstract filter

The bottom–up phase provides the approximated success patterns of every clause. If some atom does not participate in them, it need not be considered in the query evaluation since it must not participate in some real success patterns. The top–down phase finds only the approximated success patterns relevant to a given query. If some fact does not participate in them, it need not be considered in the query evaluation, since it is not relevant to a given query even if it participates in some success patterns.

We now define abstract filters using the result of the abstract analysis. The set of all the abstract instances of a rule δ in $lfp(\mathscr{D}_P^{\mathscr{A}})$ is denoted by $lfp(\mathscr{D}_P^{\mathscr{A}})[\delta]$ and the set of all the abstract instances of the ith body atom in $lfp(\mathscr{D}_P^{\mathscr{A}})[\delta]$ by $lfp(\mathscr{D}_P^{\mathscr{A}})[\delta][i]$.

Definition 22. Let $Arg\_Var(\delta, i) = \langle P1, \ldots, Pm \rangle$. The *abstract filter* $AF_{\delta,i}$ of each port $(p, \delta)/i$ in $E_{P,R}$ is defined by

$$AF_{\delta,i} = \langle P1, \ldots, Pm \rangle \in lfp(\mathscr{D}_P^{\mathscr{A}})[\delta][i].$$

which means that an atom $A(=p(t_1, \ldots, t_m))$ or its tuple satisfies $AF_{\delta,i}$, denoted by $A \in AF_{\delta,i}$ if it is in $\gamma(lfp(\mathscr{D}_P^{\mathscr{A}})[\delta][i])$. We can define the abstract filters of the input ports of a given query $Q = B_1, \ldots, B_n$ using $J_{query}^{\mathscr{A}}$ where

$$J_{query}^{\mathscr{A}} = \left\{ \alpha_c(Q\vartheta^{\mathscr{A}}) \middle| \begin{array}{l} \{B_1^{\mathscr{A}}, \ldots, B_n^{\mathscr{A}}\} \subseteq lfp(T_P^{\mathscr{A}}),\\ \vartheta^{\mathscr{A}} = mgu^{\mathscr{A}}((B_1, \ldots, B_n), (B_1^{\mathscr{A}}, \ldots, B_n^{\mathscr{A}})) \end{array} \right\}.$$

Example 23. The abstract filters with the depth of abstraction 2 for the program P in Example 16 are computed using $lfp(\mathscr{D}_P^{\mathscr{A}})$.

$$AF_{\beta,1} = \langle Arc1,\ Arc2 \rangle \in \{arc(a, c),\ arc(c, d),\ arc(d, f)\},$$
$$AF_{\beta,2} = \langle Path1,\ Path2 \rangle \in \{path(f, [f]),\ path(d, [d|-]),\ path(c, [c|-])\}.$$

The evaluation with the abstract filters generates $path(d, [d, f])$, $path(c, [c, d, f])$ and $path(a, [a, c, d, f])$ by the rule β. The static filters can not give any restrictions to the input ports of β, so the rule β generates 7 facts. If the query is $path(X, [a, c, d, f])$, the abstract filter is the same as above, and the static filters of β are $SF_{\beta,1} = Arc1 \in \{a, c, d, f\}$ and $SF_{\beta,2} = Path2 \in \{[c, d, f], [d, f], [f], []\}$. The two evaluations generate 3 facts by β, but β receives irrelevant facts $arc(a, b)$ and $arc(c, b)$ with the static filters.

Let T_P^{AF} be the operator defined for the abstract filter AF as in Definition 5. The completeness of the abstract filter is shown as follows.

Theorem 24. *$lfp(T_P^{AF})$ is equivalent to $lfp(T_P)$ with respect to the query Q.*

Proof. Straightforward from Theorem 21. □

The abstract filters are shown to be at least as powerful as the static filters, assuming both are computed using the depth-k abstraction, Practically, the abstract filter can be used in conjunction with the static filter with higher depth-k signature, because the depth-k signature is more powerful, and the computation of the static filters is less time consuming. The *conjuncted filter CF* can be defined as $CF = SF \wedge AF$. Its completeness can be easily shown from the completeness of the two filters.

Theorem 25. *$lfp(T_P^{AF}) \subseteq lfp(T_P^{SF})$ for any program P.*

Proof. We show that the body atoms of all abstract clauses in $lfp(\mathscr{D}_P^{\mathscr{A}}))$ satisfy the corresponding static filters. As $lfp(\mathscr{D}_P^{\mathscr{A}})$ computes the set of all abstract clauses in every abstract proof tree [3], the proof is by induction on the level of abstract proof tree using the safeness of SF. See [3] for details. □

6. Discussion and conclusion

Enumeration of all success patterns in the bottom–up phase may be time consuming. This problem is partly overcome by using the depth-k abstraction. Consider the rules in Example 16 and suppose a graph $G = (V, E)$ is represented by facts in the program. When the depth of abstraction is one or two, the number of instances of $path(X, [X \mid P])$ in $lfp_a(U_P^{\mathscr{A}})$ is bounded by $|V|$ and that of the rule β in $lfp_c(U_P^{\mathscr{A}})$ by $|E|$ whether the graph is acyclic or not. On the other hand, when the graph is acyclic, the number of instances of $path(X, [X \mid P])$ in $lfp(T_P)$ is bounded by $(|V|)!$, and may be infinite when it is cyclic. Moreover, small depth of abstraction is very useful as shown in the example. The depth of abstraction can be chosen depending on the property of programs, and in case the depth k should be very small, we can use the abstract filter in conjunction with the static filter with higher depth-k signature. Even in this case, the abstract filter is useful, since it is computed from the approximated success patterns.

Moreover, the proposed approach has merit in the sense that the result of the bottom–up phase can be utilized for any queries to the same program. Since many different queries are usually asked over the same program, it is a good property that the bottom–up phase is done once for a program and different top–down phases are done for different given queries. The bottom–up phase can be optimized, for example, if it is executed over the transformed program by Magic Sets. It, however, must be repeated whenever new queries are given for the same program.

In summary, we have proposed a new filter computation method using the result of the two-phase analysis. The new abstract filter is shown to be at least as powerful as the static filter under the

assumption. Practically, the abstract filter can be used in conjunction with the static filter. We are generalizing our approach to some safe approximation such as ground dependencies, sharing etc.

Acknowledgement

We are grateful to Professor R. Giacobazzi at Università di Pisa for valuable discussions. This paper is revised by his comments and suggestions.

References

[1] F. Bancilhon, D. Maier, Y. Sagiv and J.D. Ullman, Magic sets and other strange ways to implement logic programs, in: *Proc. ACM SIGACT–SIGMOD Symp. on Principles of Database Systems* (1986) 1–15.

[2] R. Barbuti, R. Giacobazzi and G. Levi, A general framework for semantics-based bottom-up abstract interpretation of logic programs, *ACM Trans. Programming Language Systems* **15** (1993) 133–181.

[3] B.-M. Chang, K.-M. Choe and T. Han, A new two-phase abstract interpretation of logic programs, Tech. Rept. CS-TR-92-73, Dept. of Computer Science, KAIST, 1992.

[4] P. Cousot and R. Cousot, Abstract interpretations: A unified lattice model for static analysis of programs by construction or approximation of fixed points, in: *Proc. Fourth ACM Symp. on Principles of Programming Languages* (1977) 238–252.

[5] M. Falaschi, G. Levi, C. Palamidessi and M. Martelli, Declarative modeling of the operational behavior of logic languages, *Theoret. Comput. Sci.* **69** (1989) 289–318.

[6] M. Kifer and E.L. Lozinskii, A framework for an efficient implementation of deductive databases, in: *Proc. 6th Advanced Database Symp.* (1986) 109–116.

[7] M. Kifer and E.L. Lozinskii SYGRAF: Implementing logic programs in a database style, *IEEE Trans. Software Engineering* **14** (1988) 922–935.

[8] K. Marriott and H. Søndergaard, Bottom-up abstract interpretation of logic programs, in: *Proc. 1988 Internat. Conf. and Symp. on Logic Programming* (1988) 733–748.

[9] D. Sacca and C. Zaniolo, The generalized counting method for recursive logic queries, in: *Proc. Internat. Conf. on Database Theory* (1986) 31–53.

[10] T. Sato and H. Tamaki, Enumeration of success patterns in logic programs, *Theoret. Comput. Sci.* **34** (1984) 227–240.