On the Optimization of Functional Symbol-Free Logic Programs

Suren Khachatryan
Yerevan State University
Yerevan, Armenia
e-mail: suren1525@gmail.com

ABSTRACT

In this paper logic programs that do not use functional symbols are studied.
Logic programs are equivalent if the sets of goals that are logical consequences of the programs are the same. The later is called Δ-equivalence.
A program is terminating if the SLD-tree, corresponding to that program and any permitted goal, is finite. In other words, for terminating programs and permitted goals, any interpreter based on SLD-resolution can decide whether a given goal is a logical consequence of a given program or not.
In general, functional symbol-free programs are not terminating.
We present a transforming algorithm by which any functional symbol-free program is transformed into a terminating program. The program obtained via transformation and the original program are Δ-equivalent.
This transformation saves the main structure of the program by adding constraints.

Keywords

Logic programming, transformation, termination, functional symbol-free programs.

1. INTRODUCTION

In logic programming Herbrand interpretation has commonly been considered. Following this convention, we consider Herbrand interpretations only.
It follows from Churchs theorem that there are no algorithms for arbitrary logic programs to resolve whether a given goal is their logical consequence or not. Thus, it seems appropriate to study specific cases of logic programs.
Some classes of terminating programs (for terminating programs, any interpreter based on SLD-resolution can decide whether a given goal is a logical consequence of a given program or not) are studied in [1,2]. For terminating programs, the SLD-tree, corresponding to a given program and a given permitted goal, is finite.
In [3] variable-free programs (i.e. programs that do not use variables) are analyzed. The given article presents a transformation by which any variable-free program is altered into a terminating program. We study logic programs which use variables but do not use functional symbols of arity 1 (henceforth referred to as FSF programs).
A decidable interpreter is known to exist for FSF programs [see 4]. However, on the whole FSF programs are not terminating.
We present a transforming algorithm by which any FSF program is transformed into a terminating program. The program obtained via transformation and the original program are Δ-equivalent.

2. PRELIMINARIES

In this section we recall some basic notations of logic programming. For notations not defined here, the reader may refer to [5,6].

We will usually denote variables by $x, y, z, ...$, functional symbols by $f, g, ...$, predicate symbols by $p, q, ...$, terms by $s, t, ...$ and formulas by $F, G, ...$. Each functional symbol has an associated non-negative integer called its arity. If the arity of a functional symbol is n, the functional symbol is called n-ary. Nullary(0-ary) functional symbols are called *constants*.

A *program clause* is a formula of the form

$$\forall(B_1 \wedge ... \wedge B_m \rightarrow A),$$

where $m \geq 0, B_1, ..., B_m$ and A are atoms.
We will write the above clause in the casual notation as

$$A : -B_1, ..., B_m.$$

The atom A is called the head and the sequence of atoms $B_1, ..., B_m$ the body of the clause.
A *goal* is a formula of the form

$$\exists(C_1 \wedge ... \wedge C_k),$$

where $k > 0$ and $C_1, ..., C_k$ are atoms.
We will write the above goal in the casual notation as

$$? - C_1, ..., C_k.$$

A *logic program* is a finite set of program clauses.
A *substitution* $\theta = \{t_1/x_1, ..., t_n/x_n\}$ is a finite set of pairs of a variable and a term, where t_i is distinct from x_i and the variables $x_1, ..., x_n$ are also distinct. A *simple expression* is either a term or an atom. If E is a simple expression $E\theta$ denotes the simple expression obtained from E by simultaneously replacing each occurrence of the variable x_i in E by the term $t_i(i = 1, ..., n)$. Let $\sigma = \{\tau_1/y_1, ..., \tau_m/y_m\}$ be a substitution, then the composition $\theta\sigma$ of θ and σ is the substitution obtained from the set

$$\{t_1\sigma/x_1, ..., t_n\sigma/x_n, \tau_1/y_1, ..., \tau_m/y_m\}$$

by deleting any pair $t_i\sigma/x_i$ for which $x_i = t_i\sigma$ and deleting any pair τ_j/y_j for which $y_j \in \{x_1, ..., x_n\}$. If $L = \{E_1, ..., E_n\}$ is a finite set of simple expressions and θ is a substitution, then $L\theta$ denotes the set $\{E_1\theta, ..., E_n\theta\}$.

Let L be a finite set of simple expressions. A substitution θ is called a *unifier* for L if $L\theta$ is a singleton. A unifier θ is called a *most general unifier (mgu)* for L if, for each unifier σ of L, there exists a substitution γ such that $\sigma = \theta\gamma$.
If Q is a goal of the form $? - C_1, ..., C_k$ then $Q\theta$ denotes the goal

$$? - C_1\theta, ..., C_k\theta,$$

where $k > 0$.

A term(atom) is called *ground* if it does not contain variables. The *Herbrand universe* U is the set of ground terms and the *Herbrand base* B is the set of ground atoms. A *Herbrand interpretation* $\mathcal{I}$ is a subset of the Herbrand base. The value of a closed formula F in interpretation $\mathcal{I}$ is defined as usual. For formulas F and G, the relation $F \models G$ means that every Herbrand model of F is a model of G.

We will denote a set of variables of an atom A by $Var(A)$. A set of variables of a goal Q of the form $? - C_1, ..., C_k$. we will denote by $Var(Q)$ and define as

$$Var(Q) = Var(C_1) \cup ... \cup Var(C_k).$$

For any program P and not empty goal Q a *logical semantics* $Log(P, Q)$ is defined as follows:

- If $P \not\models Q$, then $Log(P, Q) = \{no\}$
- If $P \models Q$ and $Var(Q) = \emptyset$, then $Log(P, Q) = \{yes\}$
- If $P \models Q$ and $Var(Q) = \{y_1, ..., y_s\}, s \geq 1$, then $Log(P, Q) = \{\langle t_1, ..., t_s\rangle \in U^s | P \models Q\theta,$ $\theta = \{t_1/y_1, ..., t_s/y_s\}\}$

where $t_1, .., t_s$ are terms.
Let P be a logic program and Q a goal. An *SLD-tree*, corresponding to P and Q, is a tree satisfying the following:

- Each node of the tree is a (possibly empty) goal
- The root node is Q
- Let $? - C_1, ..., C_k.(k \geq 1)$ be a node in the tree and suppose that C_m is the selected atom. Then, for each input clause $A : -B_1, ..., B_n$ such that C_m and A are unifiable with mgu θ, the node has a child

  $$? - C_1\theta, ..., C_{m-1}\theta, B_1\theta, ..., B_n\theta, C_{m+1}\theta, ..., C_k\theta.$$

- Nodes which are the empty goals have no children.

The set of predicate symbols used in logic program P we will denote as Π_P.
To each program P is corresponded the permitted set of goals and it is denoted as $\Delta(P)$. P_1 and P_2 programs are Δ-equivalents[1] if $\Delta(P_1) = \Delta(P_2)$ and for any goal $Q \in \Delta(P_1)$, $P_1 \models Q \Leftrightarrow P_2 \models Q$.

[1]The notion of Δ-equivalence is given in [7].

3. FUNCTIONAL SYMBOL-FREE PROGRAMS

In this section we study FSF programs.
For a FSF program, a goal is permitted if it does not use functional symbols of arity ≥ 1.

We will describe the algorithm of transformation conforming to which any FSF program P transforms to another program P' so that for any goal $Q \in \Delta(P)$, $Log(P, Q) = Log(P', Q)$ and the SLD-tree, corresponding to P' and Q, is finite. Note that transformed program P' is already not a FSF program.

We will say that atom A *precedes* atom B and denote it by $A \preceq B$, if there exists a substitution θ such that $A\theta = B$. It is easy to see that the relation $\preceq$ is transitive and reflexive.

We will say that atoms A and B are *congruent* and denote it by $A \equiv B$, if $A \preceq B$ and $B \preceq A$. It is easy to see that the relation $\equiv$ is an equivalence relation. Congruent atoms are considered identical.

3.1 Transformation

Definition 1. *For a clause S from logic program P we define set of atoms $T(P, S)$ as follows:*
$T(P, S) = \{A\theta \mid$ the substitution θ uses functional and predicate symbols only from program $P\}$,
where S is $A : -B_1, ..., B_m, m \geq 0$.

Lemma 1. *For a clause S from FSF program P*

$$\overline{\overline{T(P, S)}} = \prod_{j=1}^{l}(c + j),$$

where l is a count of the distinct variables in S, and c is a count of the distinct constants used in program P.

Definition 2. *For a logic program P we define set of atoms $T(P)$ as follows:*

$$T(P) = T(P, S_1) \cup ... \cup T(P, S_w),$$

where $S_1, S_2, ..., S_w$ are the clauses of P.

It is easy to see that

$$\overline{\overline{T(P)}} \leq \sum_{i=1}^{w} \overline{\overline{T(P, S_i)}}.$$

Let us denote sequence of terms $t_1, ..., t_k$ by $\bar{t}$, where $k \geq 0$.

Definition 3 (T2 transformation). *Let P be a FSF program. For program P we construct program P' as follows:*
For every clause S of the form $p(\bar{t}) : -p_1(\bar{t}_1), ..., p_n(\bar{t}_n) \in P, n \geq 0$, we define 2 clauses

1. *$Trans(S)$ is $p^T(\bar{t}, s(z)) : -p_1^T(\bar{t}_1, z), ..., p_n^T(\bar{t}_n, z)$,*
2. *$p(x_1, ..., x_k)$: $-p^T(x_1, ..., x_k, s^v(0))$,*

where $v = \overline{\overline{\overline{T(P)}}}$, the variables $z, x_1, ..., x_k$ are distinct, s is a new functional symbol, $p^T \notin \Pi_P$, k is the arity of p, and 0 is constant.

As a result of the 1st point of $T2$ transformation, for each clause of program P a new clause is defined, which has the same structure as the original one, but with the new predicate symbols having arity more by one.
Last parameters of new predicate symbols are intended to save values of the counters[2]. This last parameters are added as constraints.

As a result of 2nd point of $T2$ transformation, for each clause of program P a new rule is defined, which initializes the counter of corresponding clause.

3.2 Termination

The following theorem addresses the termination issue, by showing that the transformed program is terminating.

THEOREM 1 (TERMINATION). *Let P be a FSF program. P' is a program obtained from P by $T2$ transformation. Then, the SLD-tree, corresponding to a program P' and any goal $Q \in \Delta(P)$, is finite.*

In other words, for all permitted goals the transformed program is terminating.

3.3 Completeness

Having obtained a terminating program we need to prove that the logical semantics of the transformed program coincide with that of the original one.
We have the following result.

THEOREM 2 (COMPLETENESS). *Let P be a FSF program. P' is a program obtained from P by $T2$ transformation. Then, for any goal $Q \in \Delta(P), Log(P', Q) = Log(P, Q)$.*

In other words, the transformed program and the original one are Δ-equivalents.

REFERENCES

[1] D. Pedreschi, S. Ruggieri, Bounded Nondeterminism of Logic Programs. *Proc. of the International Conference on Logic Programming*, pages 350-364. MIT Pres, 1999.

[2] D. Pedreschi, S. Ruggieri, and J. G. Smaus, Classes of Terminating Logic Programs, *Theory and Practice of Logic Programming*, 2(3): 369-418, 2002.

[3] S. Khachatryan, On the Optimization of Variable-Free Logic Programs, *In Proceedings of CSIT 2011*, pages 50-51. Gitutyun, 2011.

[4] Nigiyan S.A., Khachoyan L.O., K Probleme Δ-ekvivalentnosti Logicheskikh Programm. *Dokladi NAN Armenii*, v. 99, 2, p. 99-103, 1999. (in Russian).

[5] J. W. Lloyd, Foundations of Logic Programming, *Springer-Verlag*, 1984.

[2]By term $s^m(0)$ is coded natural number m.

[6] U. Nilsson, J. Maluszski. Logic, Programming, and PROLOG. John Wiley & Sons, Inc., 2nd edition, 1995.

[7] Nigiyan S. A., Khachoyan L. O., Transformations of Logic Programs, *Programming and Computer Software*, Vol.23, N6, pp. 302-309, 1997.