PROCEEDINGS OF THE YEREVAN STATE UNIVERSITY

Physical and Mathematical Sciences 2012, № 1, p. 43–48

Informatics

ON FUNCTIONAL SYMBOL-FREE LOGIC PROGRAMS

L. A. HAYKAZYAN*

Chair of Programming and Information Technologies, YSU, Armenia

In the present article logic programs (both with and without negation) that do not use functional symbols are studied. Three algorithmic problems for functional symbol-free programs are investigated: the existence of a solvable interpreter, the problem of Δ-equivalence and the problem of logical equivalence. The first two problems are known to be decidable for functional symbol-free definite programs. We show that the third one is also decidable for such programs. In contrast, all three problems are shown to be undedicable for functional symbol-free general programs.

***Keywords*:** logic programming, functional symbol-free programs, algorithmic problems.

§ 1. Introduction. For basic definitions in logic programming we refer the reader to [1]. We study logic programs that do not use functional symbols of arity ≥ 1 (henceforth referred as FSF programs). For general programs (i.e. programs with negation) we use Clark's completion semantics from [2]. In logic programming it is customary to consider only Herbrand interpretation. We follow this convention and consider only Herbrand interpretations. While for formulas without built-in predicates (such as definite programs) this does not make a difference, it makes a difference for formulas with built-in predicates (such as completions of general programs).

Some of fundamental algorithmic problems for FSF programs (and indeed for any class of logic programs) are the following ones.

- Is there a solvable interpreter?
- Is Δ-equivalence decidable? (Two programs are called Δ-equivalent, if their semantics coincide for each permitted query, see [3]).
- Is logical equivalence (on Herbrand interpretations) of such programs decidable?

The first two problems are known to have positive answers for FSF definite programs (see [4]). In the present paper we show that the third one also has positive answer for such programs. We also show that in contrast to this, all three problems have negative answers for FSF general programs.

Let us introduce some notation that will be used throughout the paper. We fix a first order language L with a countable set of constant symbols and countable sets of functional and predicate symbols of each arity. Programs and queries in L (both definite and general) will be denoted by P and Q respectively (possibly with some subscripts or superscripts). This will not cause confusion since we study

* E-mail: levon@rock.com

definite and general programs in different sections. All programs are assumed to be functional symbol-free. Occasionally, we will need to refer to the language consisting of constant, functional and predicate symbols used in some well-formed formula F. In this case the language in question is denoted by L_F.

§ 2. Definite Programs. In this section we study definite programs (i.e. programs without negation). Definite programs possess some nice properties. In particular, each definite program P is satisfiable and, moreover, has a least Herbrand model M_P. This model is actually the set of ground atoms that logically follow from the program. The paper [4] characterizes least Herbrand models of definite programs through so called templates. Intuitively, a template K of M_P is a set of atoms (with certain minimality properties) such that every atom $A \in M_P$ is an instance of an atom $B \in K$. It is shown in [4], that all templates of the least model of a definite program are congruent (i.e. can be obtained from one another through variable renaming). It turns out that least Herbrand models of FSF definite programs have finite templates (see [4]). The following theorem is a direct consequence of this.

Theorem 2.1. There is an algorithm for deciding, if a definite query follows from a definite FSF program.

The technique of templates is also applicable to the problem of so called Δ-equivalence. Two definite programs P_1 and P_2 are called Δ-equivalent, if every query is a logical consequence of one program, iff it is a logical consequence of the other one. It follows that definite programs would be Δ-equivalent, iff the templates of their least models are congruent. This implies the next theorem.

Theorem 2.2. The problem of Δ-equivalence is decidable for FSF definite programs (see [4]).

Here we show that logical equivalence of FSF definite programs is also decidable.

Theorem 2.3. There is an algorithm for deciding, if two FSF definite programs are logically equivalent.

Proof. Let P_1 and P_2 be FSF definite programs. The formula $P_1 \leftrightarrow P_2$ is logically valid, iff so are $P_1 \rightarrow P_2$ and $P_2 \rightarrow P_1$. By symmetry considerations we only show how to check the validity $P_1 \rightarrow P_2$. The formula $P_1 \rightarrow P_2$ is valid, iff $P_1 \wedge \neg P_2$ is unsatisfiable. Now, P_1 is a universal formula and $\neg P_2$ is equivalent to an existential formula. Let P_1 be $\forall x_1,...,x_n F_1(x_1,...,x_n)$ and $\neg P_2$ be equivalent to $\exists y_1,...,y_m F_2(y_1,...,y_m)$, where F_1 and F_2 are quantifier free. We can assume that $x_1,\ldots,x_n$ are different to $y_1,...,y_m$, and then $P_1 \wedge \neg P_2$ is logically equivalent to $\exists y_1,...,y_m \forall x_1,...,x_n (F_2(y_1,...,y_m) \wedge \wedge F_1(x_1,...,x_n))$. But by Skolem's Theorem the latter is satisfiable, iff so is $\forall x_1,...,x_n (F_2(c_1,...,c_m) \wedge F_1(x_1,...,x_n))$, where $c_1,...,c_n$ are new constant symbols. Denote the last formula by G. Now G is a universal formula and, hence, has a Herbrand model in L_G if satisfiable. But since G does not use functional symbols, the Herbrand universe of L_G is finite and, hence, there are finitely many Herbrand interpretations. So, we can check all Herbrand interpretations and see, if any of them satisfies G. □

§ 3. General Programs. Theorems 2.1, 2.2 and 2.3 of the previous section demonstrate that all three algorithmic problems are decidable for FSF definite programs. This means that definite logic programs become significantly less expressive, if the usage of functional symbols is forbidden. In contrast, first-order logic does not lose its expressiveness without functional symbols. So, the special form of definite clauses becomes crucial in the decidability results of the previous section.

In this section we study the above mentioned algorithmic problems for FSF general programs. We show that in contrast to definite programs all three problems are undecidable for FSF general programs.

As stated before, we employ Clark's completion semantics for general programs (see [2]). In addition we impose the restriction to Herbrand models of completion. This means that a query Q (or its negation) is considered to be a consequence of the completion comp(P) of a general program P, if Q is true in all Herbrand models of comp(P).

However, for FSF programs the restriction to Herbrand models can be simplified.

Proposition 3.1. Let F be a first-order formula without functional symbols, possibly using equality. Then F has a Herbrand model, if it has a countable model, where each constant symbol is interpreted by a distinct element.

Proof. Let M be a countable model of F. Denote by D the domain of M. Let f be a bijection between the Herbrand universe of L and D taking the constant symbols used in F into their interpretations in M. Define a Herbrand interpretation by assigning to the atom $p(t_1,...,t_n)$ the truth value of $p^M(f(t_1),...,f(t_n))$. It is routine to check that this interpretation is a model of F. □

It is shown in [5], that the expressiveness of general programs is not extended, if we allow arbitrary formulas in bodies of clauses. To make this precise, recall the way comp(P) is obtained from P. First we rewrite each clause $p(t_1,...,t_n) \leftarrow S$ in the general form $p(x_1,...,x_n) \leftarrow \exists y_1,...,y_m(t_1 = x_1 \wedge ... \wedge t_n = x_n \wedge S)$, where $x_1,...,x_n$ are new variables and $y_1,...,y_m$ are the variables of the original clause. If

$$p(x_1,\ldots,x_n) \leftarrow E_1$$
$$\vdots$$
$$p(x_1,\ldots,x_n) \leftarrow E_k$$

are all the general forms of clauses with p in their heads, then the definition of p would be the formula $\forall x_1,...,x_n(p(x_1,...,x_n) \leftrightarrow E_1 \vee ... \vee E_k)$. As usual, the empty disjunction is understood as a logical falsehood. The completion comp(P) of P is the set of definitions of predicate symbols that occur in P.

This process does not, in any way, depend on the bodies of programs being conjunctions of literals. It is tempting to generalize general programs even further by allowing arbitrary formulas in their bodies. However, as shown in [5], given such a program P, we can construct a general program P', such that comp(P') is a conservative extension of comp(P). Recall the construction presented in [5]. Let F be a formula of first-order logic without equality. We can assume that F is built using $\neg, \vee$ and $\exists$. Let A be an atom, whose variables are different to bound variables of F. Transform the clause $A \leftarrow F$ using the following rules:

- Replace $A \leftarrow F_1 \vee F_2$ by two clauses $A \leftarrow F_1$ and $A \leftarrow F_2$.
- Replace $A \leftarrow \exists x F$ by $A \leftarrow F$.
- Replace $A \leftarrow \neg F$ by two clauses $A \leftarrow \neg p(x_1,...,x_n)$ and $p(x_1,\ldots, x_n) \leftarrow F$, where $x_1,\ldots, x_n$ are the free variables of F and p is a new predicate symbol.

Let P be a program with arbitrary formulas in bodies of clauses, and let P' be obtained from P by repeatedly applying the above transformation. It is shown in [5], that each model of comp(P') is a model of comp(P) and further each model of comp(P') is obtained from a model of comp(P) by uniquely defining interpretations

of extra predicate symbols. We will refer to this transformation as Lloyd-Topor transformation.

It is widely known that completions of general programs may be inconsistent. For example, the completion of the program consisting of the single clause $p(x) \leftarrow \neg p(x)$ is the formula $\forall x(p(x) \leftrightarrow \neg p(x))$, which does not have a Herbrand model (or any other model). Such programs present a little interest, since any query is a consequence of their completions. The next theorem shows that there is no algorithm for deciding this property of FSF general programs.

Theorem 3.1. There is no algorithm for deciding, if comp(P) has a Herbrand model for a given FSF general program P.

Proof. Let F be a closed formula of pure first-order logic (i.e. without functional and constant symbols and without equality). Let p be a new predicate symbol and a be a new constant symbol. Denote by P the general program obtained from $\{p(a) \leftarrow F \vee \neg p(a)\} \cup \{q(x_1,...,x_n) \leftarrow q(x_1,...,x_n)$: q is n-ary predicate symbol used in $F\}$ by Lloyd-Topor transformation. Then comp(P) has a Herbrand model $\Leftrightarrow$ comp(P) has a countable model (since comp(P) does not use functional and constant symbols other than a) $\Leftrightarrow$ $p(a) \leftrightarrow (F \vee \neg p(a))$ has a countable model (by Lloyd-Topor transformation) $\Leftrightarrow$ $p(a) \wedge F$ has a countable model (since the latter is logically equivalent to the former) $\Leftrightarrow F$ has a countable model (since F does not use p and a) $\Leftrightarrow F$ is consistent (by Löwenheim–Skolem Theorem). Thus, by having an algorithm for deciding, if comp(P) has a Herbrand model, we can decide pure first-order logic. But this is not possible by Church's Theorem. □

This result in itself guaranties that it is algorithmically impossible to decide, if a query is a consequence of the completion of a FSF general program. Indeed, comp(P) would have a Herbrand model, iff the query $p(a) \wedge \neg p(a)$ is not a logical consequence of it. However, this kind of undecidability is not very practical, since in practice one often writes programs with consistent completions. Thus, one can pose the question of deciding, if a query (or the negation of a query) is a conesquence of the completion of FSF general program with an additional assumption that the completion is consistent. However, this problem also turns out to be undecidable.

Theorem 3.2. There is no algorithm that takes as input FSF general program P and a general query Q with the following properties:

- if comp(P) has a Herbrand model and $\text{comp}(P) \models Q$ holds on Herbrand interpretations, then the algorithm halts with the answer "Yes";
- if comp(P) has a Herbrand model and $\text{comp}(P) \not\models Q$ holds on Herbrand interpretations, then the algorithm halts with the answer "No".

Proof. Consider Robinson arithmetic [6], which is a finitely axiomatizable and Σ_1-complete* subtheory of Peano Arithmetic in the language with equality, constant symbol 0 and functional symbols s, + and $\times$ for the successor, addition and multiplication respectively. Denote by R the conjunction of axioms of Robinson arithmetic**. Let $F(x)$ denote the Σ_1 formula representing the predicate "the Turing machine with number x halts on its number". Since R is Σ_1-complete, the Turing

---

* Σ_1-completness means that every true Σ_1 formula is provable. The formula is called Σ_1, if there is only one unbounded $\exists$ quantifier. Bounded quantifier are quantifiers of the form $\exists x \le y$ and $\forall x \le y$.

** Robinson arithmetic is often denoted by Q, but we have reserved this letter for queries.

machine with number n halts on its number, iff $R|=F(s^n(0))$. Now we want to encode R and F as a functional symbol-free general program.

In Lloyd-Topor transformation bodies of clauses are not allowed to contain equality. But this is easy to workaround. We substitute atoms $t_1=t_2$ with $eq(t_1,t_2)$, where eq is a new binary predicate symbol. Later we will add to the program the clause $eq(x,x)$. Then the completion of the program will contain $\forall y,z(eq(y,z) \leftrightarrow$ $\leftrightarrow \exists x(x=y \wedge x=z))$, which means that eq is interpreted as the equality. Next to get rid of functional symbols we use a well-known technique of substituting functional symbols with predicate symbols representing their graphs. For an n-ary functional symbol f we peak a new $n+1$-ary predicate symbol p_f, which satisfies

$$\forall x_1,...,x_n \exists y(p_f(x_1,...,x_n,y)),$$

$$\forall x_1,...,x_n,y,z(p_f(x_1,...,x_n,y) \wedge p_f(x_1,...,x_n,z) \rightarrow eq(y,z)).$$

Denote the conjunction of these two formulas as $\mathrm{Def}(f)$. Then we can replace atoms $A(f(t_1,...,t_n))$ with $\exists x(p_f(t_1,...,t_n,x) \wedge A(x))$ until no functional symbol is left.

Denote by R' and F' the formulas obtained from R and F respectively, by replacing equality with eq and functional symbols s, + and × with p_s, p_+ and $p_\times$. Let Def(R) denote the conjunction of Def(p_s), Def(p_+) and $\mathrm{Def}(p_\times)$. Then, the Turing machine with number n halts on its number, iff $R' \wedge \mathrm{Def}(R) \wedge \forall x,y(eq(x,y) \leftrightarrow x=$ $=y)| = \exists x_1,...,x_n(p_s(0,x) \wedge ... \wedge p_s(x_{n-1},x_n) \wedge F(x_n))$.

Let p and q be new unary predicate symbols and P be obtained from

$$p(0) \leftarrow (R' \wedge \mathrm{Def}(R)) \vee \neg p(0),$$

$$q(x) \leftarrow F'(x),$$

$$eq(x,x) \leftarrow,$$

$$p_s(x,y) \leftarrow p_s(x,y),$$

$$p_+(x,y,z) \leftarrow p_+(x,y,z),$$

$$p_\times(x,y,z) \leftarrow p_\times(x,y,z)$$

by Lloyd-Topor transformation. Then comp(P) is a conservative extension of $p(0) \wedge R' \wedge \mathrm{Def}(R) \wedge \forall x(q(x) \leftrightarrow F'(x)) \wedge \forall x,y(eq(x,y) \leftrightarrow x=y)$. It is clear that comp($P$) has a model (natural numbers with appropriate interpretations of predicate symbols) and even a Herbrand model (since this model is countable and there is only one constant symbol used). Let Q denote the query

$$\exists x_1,...,x_n(p_s(0,x_1) \wedge ... \wedge p_s(x_{n-1},x_n) \wedge q(x_n)).$$

Now, if the Turing machine with number n halts on its number, then $\mathrm{comp}(P) \models Q$ on all models and, in particular, on Herbrand models. On the other hand, if the Turing machine with number n does not halt on its number, then the set of natural numbers (with appropriate interpretations of predicate symbols) is a model of $\mathrm{comp}(P) \wedge \neg Q$. But then $\mathrm{comp}(P) \wedge \neg Q$ has a Herbrand model, since there is only one constant symbol used. It follows that having an algorithm with desired properties we can solve the halting problem, which is impossible. □

Corollary 1 from Theorem 3.2. There is no algorithm that takes as input FSF general program P and a general query Q with the following properties:

- if comp(P) has a Herbrand model and $\mathrm{comp}(P) \models \neg Q$ holds on Herbrand interpretations, then the algorithm halts with the answer "Yes";

• if comp(P) has a Herbrand model and $\mathrm{comp}(P) \not\models \neg Q$ holds on Herbrand interpretations, then the algorithm halts with the answer "No".

Proof. Let P be a FSF general program, whose completion has a Herbrand model and Q be a general query. Denote by P' the program obtained from P by adding the clause $p(a) \leftarrow Q$, where p is a new predicate symbol and a is a new constant symbol. Denote by Q' the query $\neg p(a)$. It is easy to see that comp(P') has a Herbrand model and $\mathrm{comp}(P') \models \neg Q'$, iff $\mathrm{comp}(P) \models Q$ (on Herbrand as well as on all interpretations). □

Corollary 2 from Theorem 3.2. There is no algorithm that takes as input FSF general program P and a general query Q with the following properties:

• if comp(P) has a Herbrand model and $\mathrm{comp}(P) \models Q$ holds on Herbrand interpretations, then the algorithm halts with the answer "Yes";

• if comp(P) has a Herbrand model and $\mathrm{comp}(P) \models \neg Q$ holds on Herbrand interpretations, then the algorithm halts with the answer "No";

• if comp(P) has a Herbrand model and none of the above holds, then the algorithm halts with the answer "Undefined".

General programs P_1 and P_2 are called Δ-equivalent, if for every query Q we have $\mathrm{comp}(P_1) \models Q \Leftrightarrow \mathrm{comp}(P_2) \models Q$ and $\mathrm{comp}(P_1) \models \neg Q \Leftrightarrow \mathrm{comp}(P_2) \models \neg Q$ (the relation $\models$ is restricted to Herbrand interpretations).

Using Theorem 3.2, it is easy to prove that neither Δ-equivalence nor logical equivalence of completions (on Herbrand interpretations) is decidable for FSF general programs (even for programs, whose completions have Herbrand models). Indeed, given a general program P and a general query Q, define programs P_1 and P_2 by adding to P clauses $p(a) \leftarrow$ and $p(a) \leftarrow Q$ respectively, where p is a new predicate symbol and a is a new constant symbol. Clearly, $\mathrm{comp}(P_1)$ and $\mathrm{comp}(P_2)$ are consistent, if so is comp(P). It is easy to see that $\mathrm{comp}(P) \models Q$ on Herbrand interpretations, iff $\mathrm{comp}(P_1)$ and $\mathrm{comp}(P_2)$ are Δ-equivalent, iff $\mathrm{comp}(P_1)$ and $\mathrm{comp}(P_2)$ are logically equivalent on Herbrand interpretations. As a corollary we have the following two Theorems.

Theorem 3.3. There is no algorithm for deciding Δ-equivalence of FSF general programs (even for programs, whose completions have Herbrand models).

Theorem 3.4. There is no algorithm for deciding logical equivalence (on Herbrand interpretations) of completions of FSF general programs (even for programs, whose completions have Herbrand models).

Received 26.09.2011

REFERENCES

1. **Lloyd J.** Foundations of Logic Programming. Springer-Verlag, 1984, 124 p.
2. **Clark K.L.** Negation as Failure. In Gallaire H. and Minker J. (eds). Logic and Data Bases. New York: Plenum Press, 1978, p. 292–322.
3. **Nigiyan S.A., Khachoyan L.O.** // Programming and Computer Software, 1997, v. 23, p. 302–309.
4. **Nigiyan S.A., Khachoyan L.O.** // Dokladi NAN Armenii, 1999, v. 99, № 2, p. 99–103 (in Russian).
5. **Lloyd J., Topor R.** // Journal of Logic Programming, 1984, v. 1, p. 225–240.
6. **Robinson R.M.** Proceedings of International Congress of Mathematics, 1950, p. 729–730.