PROCEEDINGS OF THE YEREVAN STATE UNIVERSITY

Physical and Mathematical Sciences 2011, № 2, p. 50–54

Informatics

DECIDABILITY OF Δ-EQUIVALENCE PROBLEM FOR MONADIC LOGIC PROGRAMS

L. A. HAYKAZYAN*

Chair of Programming and Information Technologies, YSU

In the present paper the Δ-equivalence problem of monadic logic programs (logic programs using only monadic functional and predicate symbols) is investigated. It is shown that contrary to the general case, the relation of Δ-equivalence is decidable in case of monadic programs. Our proof is based on the decidability of Rabin's monadic second order logic of successor functions.

***Keywords*:** logic programming, Δ-equivalence.

Introduction. The present article deals with the study of equivalence of logic programs. One of the reasons for studying the equivalence is the program transformations. Indeed, it is often desirable to optimise a program and the key requirement is that the resulting program be equivalent to the original one. The first question to ask is how to define the equivalence of logic programs. Logic programs are first order logic formulas and it would be natural to consider them as equivalent, if they are equivalent as logic formulas. This is however much stronger than is usually needed. A more useful definition would consider the programs to be equivalent, if the sets of queries that are logical consequences of programs are the same. This is the notion of the so-called Δ-equivalence. It is known that the problem of Δ-equivalence is not decidable [1]. In the present article we show the decidability of the problem of Δ-equivalence for monadic logic programs (i.e. programs containing only monadic functional and predicate symbols). Our proof is based on the decidability of Rabin's monadic second order logic of successor functions. It is also possible to show the decidability using the result that the least Herbrand model of a monadic program is a regular set [2].

Notation and Background. Let L be a first order language with at least one constant symbol. We use propositional connectives $\neg, \wedge, \vee, \rightarrow$ and quantifiers $\forall, \exists$. A *clause* is a formula of the form $\forall(C_1 \vee ... \vee C_n)$, where C_i is either an atomic formula (positive literal) or a negation of an atomic formula (negative literal). We will write the clause $\forall(A_1 \vee ... \vee A_n \vee \neg B_1 \vee ... \vee \neg B_m)$ in the form $A_1, ..., A_n \leftarrow B_1, ..., B_m$ for atomic formulas $A_1, ..., A_n$ and $B_1, ..., B_m$. In case of $m = 0$ the arrow is usually omitted. A clause with a single positive literal is called a

* E-mail: levon@rock.com

definite or *program* clause. The positive literal of a definite clause is called its *head* and negative ones – its *body*. A *(logic) program* is a non-empty finite set of definite clauses. A *query* is a formula of the form $\exists(A_1 \wedge ... \wedge A_n)$, where $A_1,...,A_n$ are atomic formulas. Note that the negation of the above query is logically equivalent to the clause $\leftarrow A_1,...,A_n$.

Definition 1. Two programs P_1 and P_2 are called Δ-equivalent (denoted as $P_1 \overset{\Delta}{\sim} P_2$), if for every query Q of L the following holds: Q is a logical consequence of P_1, iff it is a logical consequence of P_2 (see [1]).

Note that Δ-equivalent programs need not be logically equivalent (i.e. have the same models). For example, if predicate symbols p and q never occur in a program, then the addition of clause $p(x) \leftarrow q(x)$ does not allow us to infer any new queries, so the resulting program would be Δ-equivalent (but not logically equivalent) to the original one. Another observation to make is that the definition of Δ-equivalence depends on the underlying first order language. For example, the question whether the programs $p(x)$ and

$$p(f(x)) \leftarrow p(x),$$
$$p(a)$$

are Δ-equivalent or not depends upon whether the underlying language contains any functional or constant symbols besides f and a.

The *Herbrand universe* of L (denoted as U_L) is the set of all ground terms (terms not containing variables), and *Herbrand base* of L (denoted as B_L) is the set of all ground atoms. An interpretation I is called a *Herbrand interpretation*, if

- the domain of the interpretation is U_L (U_L is not empty as L contains at least one constant);
- every constant symbol corresponds to itself;
- every functional symbol f corresponds to a function f^I such that $f^I(t_1,...,t_n) = f(t_1,...,t_n)$ for every $t_1,...,t_n \in U_L$.

A Herbrand interpretation corresponds to a subset of B_L (the set of atoms true under the interpretation). Conversely, every subset of B_L defines a Herbrand interpretation. We will identify Herbrand interpretations with subsets of B_L.

Fact 1. If a set of clauses has a model, then it has a Herbrand model (see [3]).

Fact 2. If A is a non-empty set and M_α is a Herbrand model of a program P for every $\alpha \in A$, then $\bigcap_{\alpha \in A} M_\alpha$ is also a model of P (see [3]).

Thus, each program P would have the *least Herbrand model* M_P that is the intersection of all its Herbrand models (the set of Herbrand models of a program is definitely not empty).

A *substitution* is a finite set of pairs $\theta = \{x_1 / t_1,...,x_n / t_n\}$, where x_i are distinct variables and t_i are terms for $i = 1,...,n$, $n \geq 0$. If F is a formula and θ is the above substitution, then $F\theta$ denotes the formula, obtained from F by

simultanious substitution of all free occurrences of x_i by t_i. $F\theta$ is then called an *instance* of F. From the above discussion it is easy to prove the following

Fact 3. A query $\exists(A_1 \wedge ... \wedge A_n)$ is a logical consequence of program P, iff for some substitution θ we have $A_i\theta \in M_p$ for $i = 1,...,n$ (see [3]).

Corollary 1. Programs P_1 and P_2 are Δ-equivalent, iff $M_{P_1} = M_{P_2}$.

Let $g(P)$ be the set of ground instances of clauses of P. Define a function $T_P : 2^{B_L} \to 2^{B_L}$ as follows $T_P(I) = \{A_0 : A_0 \leftarrow A_1,...,A_n \in g(P)$ and $A_1,...,A_n \in I\}$.

For an ordinal α define $T_P \uparrow \alpha$ as:

- $T_P \uparrow 0 = \emptyset$;
- $T_P \uparrow (\beta + 1) = T_P(T_P \uparrow \beta)$;
- $T_P \uparrow \delta = \bigcup_{\beta<\delta} T_P \uparrow \beta$ for a limit ordinal δ.

Fact 4. For a program P, $T_P \uparrow \omega = M_P$ (see [3]).

Let us introduce Rabin's monadic second order logic of successor functions. Fix some natural number $n \geq 1$. The theory *SnS* consists of a countable set of object variables, a countable set of predicate variables, a single constant symbol a and n monadic functional symbols $r_1,...,r_n$. The *Terms* of *SnS* are exactly those of the first order logic constructed from $a, r_1,...,r_n$ and object variables. The set of *formulas* of *SnS* is defined as the least set such that

- if X is a predicate variable and t is a term, then $X(t)$ is a formula;
- if F and G are formulas, x is an object variable and X is a predicate variable, then $(\neg F),(F \wedge G),(F \vee G),(F \to G),(\forall xF),(\exists xF),(\forall XF),(\exists XF)$ are formulas.

The semantics of *SnS* formulas is defined with respect to the Herbrand interpretation (second order quantifiers range over subsets of the Herbrand universe). Observe that since *SnS* does not contain predicate symbols, there is a unique Herbrand interpretation. Thus, the closed formulas of *SnS* are either true or false.

Fact 5. There is an algorithm for deciding whether a given closed *SnS* formula is true or false (see [4]).

Results and Discussions. It is known that the problem of Δ-equivalence of logic programs is not decidable, i.e. there is no algorithm to decides whether two given programs are Δ-equivalent in the given language or not [1]. Here we study the decidability of Δ-equivalence under the additional assumption that the language in question is monadic (i.e. predicate and functional symbols of the language are monadic).

First observe that we can further assume that the first order language is finite. Indeed, let L be a first order language with countable number of constant symbols $c_1, c_2,...$ and consider programs P_1 and P_2. Let $c_1,...,c_n$ be the constant symbols used in P_1 and P_2. From symmetry considerations it is clear that the ground atom A is a logical consequence of a program P_i, iff the atom A', obtained from A by replacing all constant symbols c_j with $j > n$ by c_{n+1}, is a logical consequnce of the program. Thus, P_1 and P_2 would be Δ-equivalent in L, iff they are Δ-equivalent in a language obtained from L by dropping all constant symbols

c_j with $j > n+1$. The case of a language with countable number of predicate or functional symbols is handled similarly. Hereafter L will denote a first order finite monadic language with at least one constant symbol.

A program is said to be in *canonical* form, if every variable in the head of every clause (in case of monadic programs there may be at most one variable in the head of a clause) occurs also in its body. We prove that every program has a Δ-equivalent canonical form.

Proposition 1. For every monadic program P in L there is a program P' in canonical form such that $P \overset{\Delta}{\sim} P'$.

Proof. Assume that P contains a clause $A_0(x) \leftarrow A_1,...,A_n$, where the variable *x* occurs in A_0, but not in $A_1,...,A_n$. The clause is removed from *P* and instead the following ones are added: for every constant *c* of L the clause $A_0(c) \leftarrow A_1,...,A_n$ and for every functional symbol *f* of *L* the clause $A_0(f(x)) \leftarrow A_0(x),\ A_1,...,A_n$ (it is essential that the language is finite). Using Fact 4 and Corollary 1 it is easy to prove that the resulting program is Δ-equivalent to P in L. Repeating this procedure for every clause that violates the definition of a canonical program, one would come to a program P' that is in canonical form and is Δ-equivalent to P in L. □

Thus, we can only study Δ-equivalence problem for canonical programs. Let L be a monadic language and P a canonical program. We are going to define a monadic language L' with a single constant symbol and a program P' in it, which will be in a sense equivalent to P. The first order language is defined by its constant, functional and predicate symbols:

- the single constant symbol of L' is denoted by a;
- functional symbols of L' are those of L together with a new unary functional symbol f_c for each constant symbol c of L;
- predicate symbols of L' are precisely those of L.

A ground term t' in L' of the form $f_1(...f_n(f_c(a))...)$, where $f_1,...,f_n$ are functional and c is a constant symbol of L, is called *normal*. We associate the term $t = f_1(...f_n(c)...)$ of L to the above normal term. It is easy to see that this association is a one-to-one mapping between ground terms of L and normal terms of L'. The program P' is obtained from P by replacing each ground term t with the corresponding normal term t'.

Lemma 1. A ground atom $p(t')$ is a logical consequence of P', iff t' is normal and $p(t)$ is a logical consequence of P.

Proof. Let us prove by induction on n that $p(t') \in T_{P'} \uparrow n$, iff t' is normal and $p(t) \in T_P \uparrow n$. The case of $n = 0$ is obvious.

Assume the hypothesis holds for n, let us prove it for $n+1$. Let $p(t') \in T_{P'} \uparrow n+1$. Then there is a clause $A'_0 \leftarrow A'_1,...,A'_m$ in P' and a ground substitution $\theta' = \{x_1 / t'_1,...,x_k / t'_k\}$ of its variables such that $A'_i\theta' \in T_{P'} \uparrow n$ for $i = 1,...,m$ and $A'_0\theta' = p(t')$. By induction hypothesis $A'_i\theta' = q_i(s'_i)$, where s'_i is normal and $q_i(s_i) \in T_P \uparrow n$. Since P and, hence, P' are in canonical form, all the variables $x_1,...,x_k$ occur in the body of the clause. This implies that $t'_1,...,t'_k$ are all normal

and, hence, so is t'. Let $A_0 \leftarrow A_1,...,A_m$ be the clause of P from, which $A_0' \leftarrow A_1',...,A_m'$ is derived and $\theta = \{x_1 / t_1,...,x_k / t_k\}$. Clearly $A_i\theta = q_i(s_i) \in T_P \uparrow n$ and so $A_0\theta = p(t) \in T_P \uparrow n+1$.

Conversely, assume that $p(t) \in T_P \uparrow n+1$. Then there is a clause $A_0 \leftarrow A_1,...,A_m$ and a substitution $\theta = \{x_1 / t_1,...,x_k / t_k\}$ such that $A_0\theta = p(t)$ and $A_i\theta \in T_P \uparrow n$. Similar to the other direction, for the derived clause $A_0' \leftarrow A_1',...,A_m'$ and $\theta' = \{x_1 / t_1',...,x_k / t_k'\}$ we have $A_i'\theta' \in T_{P'} \uparrow n$ and so $A_0'\theta' = p(t') \in T_{P'} \uparrow n+1$. □

Corollary 2. Let L be a finite monadic language, P_1 and P_2 – canonical programs in it. Let L', P_1' and P_2' be obtained from L, P_1 and P_2 respectively by means of the above described procedure. P_1 is Δ-equivalent to P_2 in L, iff P_1' is Δ-equivalent to P_2' in L'. □

The language L' has single constant symbol and finite number of monadic functional symbols that are interpreted as successor functions. This enables us to use the second order power of *SnS* to express statements about Herbrand interpretations of L'. In particular Δ-equivalence turns out to be expressible in *SnS*.

Theorem 1. There is an algorithm for deciding whether given programs are Δ-equivalent in a given monadic language or not.

Proof. Let L be a monadic language, P_1 and P_2 – programs in it. As was previously noted we can assume that L is finite. First we construct canonical forms P_1' and P_2' of P_1 and P_2 respectively. Clearly $P_1 \overset{\Delta}{\sim} P_2 \Leftrightarrow P_1' \overset{\Delta}{\sim} P_2'$. Next we construct a monadic language with a single constant L' and programs P_1'' and P_2'' such that $P_1' \overset{\Delta}{\sim} P_2' \Leftrightarrow P_1'' \overset{\Delta}{\sim} P_2''$, where the first equivalence is considered in L and the second one in L'.

Let a be a single constant symbol, $r_1,...,r_n$ be functional symbols and $p_1,...,p_m$ be predicate symbols of L'. The programs P_1'' and P_2'' would not be Δ-equivalent, iff for some predicate symbol p_i and some ground term t the atom $p_i(t)$ is a logical consequence of P_1'', not of P_2''. From the semantics of *SnS* it is clear that the former takes place, iff the formula $\exists t(\forall p_1...p_m(P_1'' \rightarrow p_i(t)) \wedge \exists p_1...p_m(P_2'' \wedge \neg p_i(t)))$ of *SnS* is true and the latter takes place, if the formula $\exists t(\forall p_1...p_m(P_2'' \rightarrow p_i(t)) \wedge \exists p_1...p_m(P_1'' \wedge \neg p_i(t)))$ of *SnS* is true ($p_1,...,p_m$ are predicate variables and t is an object variable).

Thus, to decide whether P_1'' and P_2'' are Δ-equivalent or not we need to check $2m$ formulas of *SnS* for m values of i. □

Received 17.09.2010

REFERENCES

1. **Nigiyan S.A., Khachoyan L.O.** Programming and Computer Software, 1997, v. 23, p. 302–309.
2. **Matos A.** Theoretical Computer Science, 1997, v. 176, p. 175–204.
3. **Lloyd J.** Foundations of Logic Programming. Springer-Verlag, 1984, 124 p.
4. **Rabin M.** Bulletin of the American Mathematical Society, 1968, v. 74, p. 1025–1029.