A Transformation Technique for Datalog Programs Based on Non-Deterministic Constructs*

Petros Potikas[1], Panos Rondogiannis[1], and Manolis Gergatsoulis[2]

[1] Department of Informatics & Telecommunications,
University of Athens, Panepistimiopolis, 15 784 Athens, Greece,
ppotik@cs.ntua.gr, prondo@di.uoa.gr
[2] Institute of Informatics & Telecommunications, N.C.S.R. 'Demokritos',
153 10 Aghia Paraskevi Attikis, Greece
manolis@iit.demokritos.gr

Abstract. Recently, a novel transformation technique for Datalog programs, called the *branching transformation*, was introduced by the authors. In this paper we propose a significant extension of the branching transformation which we believe opens-up a promising new direction of research in the area of value-propagating Datalog optimizations. One of the novel characteristics of the proposed approach is that the target language is Datalog$_{nS}$ extended with *choice predicates*, a form of non-deterministic construct that was originally introduced in the area of *intensional logic programming*.

Keywords: Logic Program Transformations, Deductive Databases.

1 Introduction

An important class of transformation techniques for Datalog programs is based on the idea of propagating the input values of the top level goal of the source program in order to restrict the generation of atoms in the bottom-up computation. Such techniques include the *counting transformation* [SZ88], the *magic sets* [BR91], the *pushdown approach* [GSZ99], and so on. A novel transformation that also belongs to this category, is the *branching transformation* [RG98,RG01], that was recently proposed by the authors for the class of chain Datalog programs. The branching transformation was inspired by a similar technique that has been proposed for functional programming languages [RW97,RW99].

In this paper, we extend the branching approach so as that it can handle a significantly broader class of well-moded Datalog programs. One of the novel characteristics of the new branching approach is that the target language is Datalog$_{nS}$ extended with choice predicates [OW94], a non-deterministic construct that was originally introduced in intensional logic programming [OW92].

* This work has been partially supported by the University of Athens under the project "Design, Implementation and Applications of Intensional Programming Languages" (grant no 70/4/5827).

The use of choice predicates allows the transformation of clauses containing multiple consumption of variables. We believe that the use of non-deterministic constructs opens-up a promising direction of research in the area of Datalog optimizations. The main contributions of the paper can be summarized as follows:

- We propose a new value-propagating transformation technique for a class of moded Datalog programs. Actually, the class of programs that we consider is broader than those considered by other related transformations.
- We demonstrate that non-deterministic constructs can prove especially useful for defining new powerful transformations for classical logic programs. In particular, we demonstrate that multiple consumption of variables in Datalog programs can be treated effectively using choice predicates.

2 An Outline of the Technique

The branching transformation was recently introduced by the authors in the area of logic programming, and more specifically for the class of chain Datalog programs [RG98,RG01]. The main idea behind the branching transformation can be illustrated by transforming the following clause:

```
p(X,Z) ← q(X,Y),r(Y,Z).
```

By transforming this clause we obtain:

```
p2(L,Z) ← r2([l2|L],Z).
r1([l2|L],Y) ← q2([l1|L],Y).
q1([l1|L],X) ← p1(L,X).
```

Intuitively, the above clauses reflect the argument passing that takes place when the initial program is executed. Notice that for every predicate in the initial program two new predicates have been introduced. Each one of the new predicates has two arguments, a control one (in the form of a list of natural numbers) and a data one which encodes the argument beeing passed.

However, the branching technique in its initial form could not be used to apply to Datalog programs in which there exist multiple consumptions of variables. The following example demonstrates the issues that arise in such a case. Consider a Datalog program that contains the clause:

```
p(X,Z) ← q(X,W),r(X,W,Z).
```

The problem with the above clause, arise from the fact that `X` appears twice in the body of the clause. When attempting to apply the branching technique to the above program, the relationship between the two different occurrences of `X` is lost, and the resulting program is no longer semantically equivalent to the initial one. More specifically, a naive translation of the above clause would produce (among others) the two following clauses:

```
q1([l1|L],X) ← p1(L,X).
r1([l2|L],X) ← p1(L,X).
```

Notice that the two occurrences of `X` in the body of the initial clause have been separated from each other as they have been placed in different clauses in the target program, and it is therefore possible for them to instantiate to different values (something which was not the case in the original clause). In this way the resulting program may produce answers which are not included in the set of answers of the initial program.

In this paper, we propose a solution to the above problem based on *choice predicates* [OW94], a form of non-deterministic construct that has been proposed in the area of *intensional logic programming* [OW92] (similar non-deterministic constructs have also been considered in other forms in [GPZ01]). Choice predicates are declarative in nature and have a well-defined and elegant semantics [OW94]. The basic idea behind choice predicates is that under a given context (represented by the list `L` in the above example), a predicate can only be true of a unique value. Therefore, the above two clauses can be instead written as:

```
q1([l1|L],X) ← #p1(L,X).
r1([l2|L],X) ← #p1(L,X).
```

where `#p1` is the choice version of the predicate `p1`, which at any given `L` can be true of only one value. This restores the connection between the two occurrences of `X`, resulting in a target program equivalent to the source one.

It should be noted that the target programs of the proposed transformation consist of clauses that contain at most one IDB atom in their body. Moreover, all clauses produced that correspond to IDB predicates of the source program, use only a single data variable.

3 The Source Language of the Transformation

The class of programs on which the proposed transformation applies is a subclass of Datalog. A *Datalog program* P consists of a finite set of function-free Horn rules. Predicates that appear in the head of some rule in P are called *IDB predicates* (IDBs), while predicates appearing only in the bodies of the rules of P are called EDB predicates (EDBs). A set D of ground facts (or unit clauses) defining the EDBs is often called a *database*. A Datalog program P together with a database D is denoted by P_D.

Definition 1. *A clause*

$$p_0(v_0, Z_n) \leftarrow p_1(v_1, Z_1), p_2(v_2, Z_2), \ldots, p_n(v_n, Z_n).$$

with $n \geq 0$*, is called* consecutive consumption clause *(or* cc-clause *for short) if:*

1. *Each* v_i*, for* $i = 0, \ldots, n$ *is a nonempty vector of distinct variables, and* $Z_1, \ldots, Z_n$ *are distinct variables.*
2. $vars(v_0) = vars(v_1)$ *and* $vars(v_i) = \{Z_{i-1}\} \cup u_{i-1}$*, for* $i = 2, \ldots, n$*, where* $u_{i-1} \subseteq vars(v_{i-1})$*.*
3. $Z_i \notin \bigcup_{j \leq i} vars(v_j)$*, for* $i = 1, \ldots, n$*.*

A program P is said to be a consecutive consumption Datalog program *(or* cc-Datalog program*) if all its clauses are cc-clauses.*

It should be mentioned here that cc-Datalog clauses are moded; the terms v_i of the above definition correspond to input arguments while each Z_i corresponds to the single output argument of each atom. An occurrence of a variable in an input argument of the head or in an output argument of an atom in the body will be called *productive*; otherwise will be called *consumptive*.

Example 1. The following clause is a cc-clause:
$\mathtt{p}(\overset{+}{\mathtt{X}},\overset{+}{\mathtt{Y}},\overset{-}{\mathtt{Z}}) \leftarrow \mathtt{q}(\overset{+}{\mathtt{Y}},\overset{+}{\mathtt{X}},\overset{-}{\mathtt{W}}),\ \mathtt{r}(\overset{+}{\mathtt{X}},\overset{+}{\mathtt{W}},\overset{-}{\mathtt{R}}),\ \mathtt{s}(\overset{+}{\mathtt{W}},\overset{+}{\mathtt{R}},\overset{-}{\mathtt{Z}}).$

The intuition behind this class is that each variable produced in an atom can only be consumed in a sequence of (one or more) consecutive atoms immediately following the atom that produced it. Many natural Datalog programs belong to this class; for example, the chain Datalog programs are members of this class.

Definition 2. *A* simple cc-Datalog program *is a cc-Datalog program in which every clause has at most two atoms in its body.*

The following proposition (whose proof is straightforward using unfold/fold transformations [TS84,GK94]) establishes the equivalence between cc-Datalog programs and simple cc-Datalog ones. Notice that by $M(p, P_D)$ we denote the set of atoms in the minimal model of P_D whose predicate symbol is p.

Proposition 1. *Every cc-Datalog program P can be transformed into a simple cc-Datalog program P' such that $M(p, P_D) = M(p, P'_D)$ for all predicate symbols appearing in P.*

4 The Target Language of the Transformation

The target language of the transformation is Datalog$_{nS}$ [Cho95] extended with choice predicates [OW94]. *Datalog$_{nS}$* is an extension of Datalog in which atoms and terms may have a single distinguished argument, which is said to be *functional*, in addition to the usual *data arguments* (the structure of the functional terms is not however arbitrary, see [Cho95] for details). Datalog$_{nS}$ has as a special case a language which is exactly like Datalog but in which the first argument of every predicate is a list of natural numbers[1]. In the rest of this paper when we refer to Datalog$_{nS}$ we will mean this particular subset.

Choice predicates [OW94] were initially introduced in the area of temporal logic programming as a means for ensuring that a given predicate is single-valued at a particular moment in time (or more generally at a particular context). To motivate choice predicates consider writing a program whose purpose is to assign

[1] In the original formulation of the branching transformation [RG01], sequences of temporal operators were used instead of lists. It can be easily shown that both formulations are equivalent.

a classroom to persons over different moments in time. The problem is to find all different such assignments in such a way that at every different moment only a single person occupies a classroom. The EDB predicate `requests_class(Time, Person)` expresses the fact that `Person` requests the particular classroom at time `Time`. The IDB predicate `uses_class(Time, Person)` expresses the fact that `Person` *actually* uses the classroom at time `Time`.

```
requests_class(0,tom).
requests_class(0,nick).
requests_class(1,mary).
uses_class(Time,Person) ← #requests_class(Time,Person).
```

In the above program, `#requests_class` is the choice predicate that corresponds to the (classical) predicate `requests_class`. The crucial property of a choice predicate is that it is single-valued for any given time-point. The above program does not have a unique minimum model (as is the case in classical logic programming); instead, it has a set of minimal models, one for every different possible (functional) assignment of persons over the moments in time. More specifically, if we denote by D the database of the above program (corresponding to the three facts), the two minimal models of the program are the following:

$$M_1 = \{\#\texttt{uses\_class}(\texttt{0},\texttt{tom}), \#\texttt{uses\_class}(\texttt{1},\texttt{mary}), \texttt{uses\_class}(\texttt{0},\texttt{tom}),\\ \texttt{uses\_class}(\texttt{1},\texttt{mary}), \#\texttt{requests\_class}(\texttt{0},\texttt{tom}),\\ \#\texttt{requests\_class}(\texttt{1},\texttt{mary})\} \cup D$$

and

$$M_2 = \{\#\texttt{uses\_class}(\texttt{0},\texttt{nick}), \#\texttt{uses\_class}(\texttt{1},\texttt{mary}), \texttt{uses\_class}(\texttt{0},\texttt{nick}),\\ \texttt{uses\_class}(\texttt{1},\texttt{mary}), \#\texttt{requests\_class}(\texttt{0},\texttt{nick}),\\ \#\texttt{requests\_class}(\texttt{1},\texttt{mary})\} \cup D$$

Choice predicates are not necessarily restricted to apply to simple temporal logic programming languages such as the one used in the above example; they are also applicable to more general intensional programming languages [OW92] that include the language Datalog$_{nS}$ which we have adopted for defining the proposed transformation. More specifically, in [OW94], Orgun and Wadge develop a general semantic framework for choice predicates that can apply to a wide class of intensional logic programming languages. As it is demonstrated in [OW94], an intensional logic program extended with choice predicates has a set of *canonical minimal models* representing the intended meaning of the transformation.

The language we will be using in the rest of this paper as the target language of the transformation, is Datalog$_{nS}$ extended with choice predicates.

Definition 3. *A* Choice Datalog$_{nS}$ *program is a Datalog$_{nS}$ program which may contain choice atoms in the bodies of its clauses.*

5 The Transformation Algorithm

In this section we provide a formal definition of the transformation algorithm. The algorithm is subsequently illustrated by a representative example.

The algorithm: Let P be a given simple cc-Datalog program and G a given goal clause. For each $(n+1)$-ary predicate p in P, we introduce $n+1$ binary IDB predicates $p_1^+, \ldots, p_n^+, p^-$ together with their choice versions $\#p_1^+, \ldots, \#p_n^+$, $\#p^-$, where p_i^+ corresponds to the i-th input argument of p and p^- to the output argument of p. The transformation processes each clause in P and the goal clause G and gives as output a Choice Datalog$_{nS}$ program P^* together with a new goal clause G^*. We assume the existence of a labeling function which assigns different labels to atoms that appear in the bodies of clauses of P. Labels are natural numbers and are denoted by $l_1, l_2, \ldots$. The algorithm is defined by a case analysis, depending on the form of the clause being processed every time:

Case 1: Let C be a clause of the form:

$$p(t_0, Z) \ \leftarrow \ q(t_1, Y), r(t_2, Z).$$

and let l_1, l_2 be the labels of $r(t_2, Z)$ and $q(t_1, Y)$ respectively. Then C is transformed in the following way:

1. The following clause is added to P^*:
$$p^-(L, Z) \leftarrow \ \#r^-([l_1|L], Z).$$
2. Let X be a variable that appears in the k-th position of t_0 and also in the m-th position of t_1. Then, the following clause is added to P^*:
$$q_m^+([l_2|L], X) \leftarrow \ \#p_k^+(L, X).$$
The case of a variable appearing in t_0 and also in t_2, is treated analogously.
3. If the output variable Y of q appears in the m-th position of t_2, then the following clause is added to P^*:
$$r_m^+([l_1|L], Y) \leftarrow \ \#q^-([l_2|L], Y).$$

Case 2: Let C be a clause of the form:

$$p(t_0, Z) \ \leftarrow \ q(t_1, Z).$$

and let l_3 be the label of $q(t_1, Z)$. Then C is transformed as follows:

1. The following clause is added to P^*:
$$p^-(L, Z) \leftarrow \ \#q^-([l_3|L], Z).$$
2. Let X be a variable that appears in the k-th position of t_0 and also in the m-th position of t_1. Then, the following clause is added to P^*:
$$q_m^+([l_3|L], X) \leftarrow \ \#p_k^+(L, X).$$

Case 3: For every EDB predicate p of P with n input variables and one output variable, a new clause of the following form is added to P^*:

$$p^-(L, Y) \leftarrow p(X_1, \ldots, X_n, Y), \#p_1^+(L, X_1), \ldots, \#p_n^+(L, X_n).$$

Case 4: The transformation of the goal clause: $\leftarrow p(a_1, \ldots, a_n, Y)$. results to a set of n new facts, which are added to P^*:

$$p_i^+([\,], a_i).$$

for $i = 1, \ldots, n$. The new goal clause G^* is: $\leftarrow \ p^-([\,], Y)$.

Example 2. Let P be the cc-Datalog program that follows, where `q`, `p` are IDB predicates and `e`, `f`, `g` are EDB predicates:

```
← q(a1,Z).
q(X,Z) ← f(X,Z).
q(X,Z) ← e(X,Y),p(Y,Z).
p(X,Z) ← q(X,W),g(X,W,Z).
```

The program (together with a new goal clause) obtained by applying the transformation algorithm to $P \cup \{\leftarrow \mathtt{q(a1,Z)}\}$ is:

```
← q2([],Z).
q1([],a1).
q2(L,Z) ← #f2([5|L],Z).
f1([5|L],X) ← #q1(L,X).
q2(L,Z) ← #p2([2|L],Z).
p1([2|L],Y) ← #e2([1|L],Y).
e1([1|L],X) ← #q1(L,X).
p2(L,Z) ← #g3([4|L],Z).
g2([4|L],W) ← #q2([3|L],W).
g1([4|L],X) ← #p1(L,X).
q1([3|L],X) ← #p1(L,X).
e2(L,Y) ← e(X,Y), #e1(L,X).
f2(L,Y) ← f(X,Y), #f1(L,X).
g3(L,Z) ← g(X,Y,Z), #g1(L,X), #g2(L,Y).
```

Consider now the following database D:

```
e(a1,a2). f(a2,a). g(a2,a,b3).
e(a1,b1). f(b1,b). g(b1,a,a3).
```

Both the initial and the final program have the single answer $\mathtt{Z} = \mathtt{b3}$. However, if we replace the choice predicates with the corresponding classical ones, then the resulting program will have an extra (incorrect) answer, namely $\mathtt{Z} = \mathtt{a3}$.

The correctness of the transformation algorithm is demonstrated by the following theorem (whose proof is based on fixpoint induction).

Theorem 1. *Let P be a simple cc-Datalog program and $\leftarrow p(a_1,\ldots,a_n,Y)$ be a goal clause. Let P^* be the Choice Datalog$_{nS}$ program obtained by applying the transformation algorithm to $P \cup \{\leftarrow p(a_1,\ldots,a_n,Y)\}$. Then $\#p^-([\,],b)$ belongs to a canonical model M of P^*_D iff $p(a_1,\ldots,a_n,b) \in T_{P_D} \uparrow \omega$.*

6 Related and Future Work

Two transformation techniques that are more closely related to the proposed transformation are the *counting method* [SZ88] and the *pushdown approach* [GSZ99]. In the counting technique integer indices are used in order to encode the structure of the computation used to generate an atom. To our knowledge, the counting approach has not been extended to apply to a class as extensive as the one presented in this paper. In the pushdown method, the relationship between chain queries, context-free languages and pushdown automata is used to rewrite the queries in a form suitable for bottom-up evaluation. The pushdown method does not apply to programs that contain multiple consumption of variables, and is therefore more restricted than our method. Finally, our algorithm also relates to techniques aiming at transforming non-linear into linear Datalog programs [AGK96], which however do not apply to the whole class of chain programs.

We believe that the work presented in this paper can be extended in various ways. We are currently investigating possible extensions of the source language as well as performance comparison with related optimization techniques.

References

[AGK96] F. Afrati, M. Gergatsoulis, and M. Katzouraki. On transformations into linear database logic programs. In D. Bjørner, M. Broy, and I. Pottosin, editors, *Perspectives of Systems Informatics (PSI'96), Proceedings*, LNCS 1181, pages 433–444. Springer-Verlag, 1996.

[BR91] C. Beeri and R. Ramakrishnan. On the power of magic. *The Journal of Logic Programming*, 10(1,2,3 & 4):255–299, 1991.

[Cho95] Jan Chomicki. Depth-bounded bottom-up evaluation of logic programs. *The Journal of Logic Programming*, 25(1):1–31, 1995.

[GK94] M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. In M. Hermenegildo and J. Penjam, editors, *Programming Language Implementation and Logic Programming (PLILP'94), Proceedings*, LNCS 844, pages 340–354. Springer-Verlag, 1994.

[GPZ01] F. Giannotti, D. Pedreschi, and C. Zaniolo. Semantics and expressive power of non-deterministic constructs in deductive databases. *Journal of Computer and Systems Sciences*, 62(1):15–42, 2001.

[GSZ99] S. Greco, D. Saccà, and C. Zaniolo. Grammars and automata to optimize chain logic queries. *International Journal on Foundations of Computer Science*, 10(3):349–372, 1999.

[OW92] M. A. Orgun and W. W. Wadge. Towards a unified theory of intensional logic programming. *The Journal of Logic Programming*, 13(4):413–440, 1992.

[OW94] M. A. Orgun and W. W. Wadge. Extending temporal logic programming with choice predicates non-determinism. *Journal of Logic and Computation*, 4(6):877–903, 1994.

[RG98] P. Rondogiannis and M. Gergatsoulis. The intensional implementation technique for chain datalog programs. In *Proc. of the 11th International Symposium on Languages for Intensional Programming (ISLIP'98), May 7-9, Palo Alto, California, USA*, pages 55–64, 1998.

[RG01] Panos Rondogiannis and Manolis Gergatsoulis. The branching-time transformation technique for chain datalog programs. *Journal of Intelligent Information Systems*, 17(1):71–94, 2001.

[RW97] P. Rondogiannis and W. W. Wadge. First-order functional languages and intensional logic. *Journal of Functional Programming*, 7(1):73–101, 1997.

[RW99] P. Rondogiannis and W. W. Wadge. Higher-Order Functional Languages and Intensional Logic. *Journal of Functional Programming*, 9(5):527–564, 1999.

[SZ88] D. Saccà and C. Zaniolo. The generalized counting method for recursive logic queries. *Theoretical Computer Science*, 4(4):187–220, 1988.

[TS84] H. Tamaki and T. Sato. Unfold/fold transformations of logic programs. In Sten-Åke Tarnlund, editor, *Proc. of the Second International Conference on Logic Programming*, pages 127–138, 1984.