Datalog with Non-deterministic Choice Computes *NDB-PTIME*

Fosca Giannotti[1], and Dino Pedreschi[2]

[1] CNUCE Institute of CNR, Via S. Maria 36, 56125 Pisa, Italy
e-mail: fosca@cnuce.cnr.it

[2] Dipartimento di Informatica, Univ. Pisa, Corso Italia 40, 56125 Pisa, Italy
e-mail: pedre@di.unipi.it

July 3, 1997

Abstract

This paper addresses the issue of non deterministic extensions of logic database languages. After providing an overview of the main proposals in the literature, we concentrate on the analysis of the *dynamic choice* construct from the point of view of the expressive power. We show how such construct is capable of expressing several interesting deterministic and non deterministic problems, such as forms of negation, and ordering. We then prove that Datalog augmented with the dynamic choice expresses exactly the non deterministic time-polynomial queries. We thus obtain a complete characterization of the expressiveness of the dynamic choice, and conversely achieve a characterization of the class of queries *NDB-PTIME* by means of a simple, declarative, and efficiently implementable language.

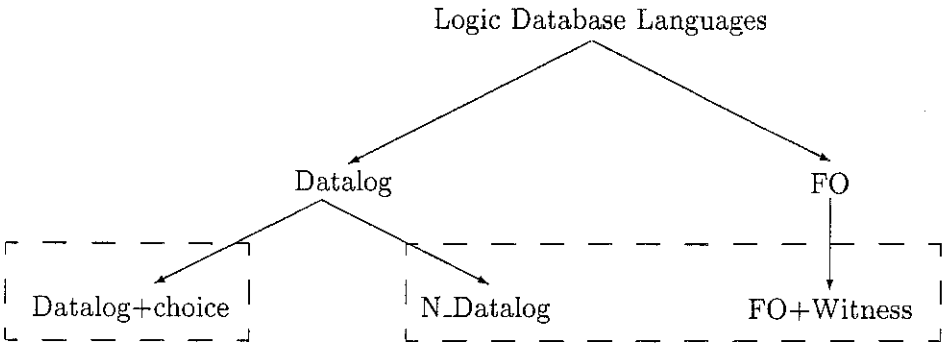
Notes. This article is an extended, revised version of [10].

1 Introduction

Two main classes of logic database languages have been proposed in the literature. One is the class of *FO* database languages, based on the relational calculus, i.e. on the first-order logic interpretation of the relational data model. The other one is the class of Datalog languages, a subset of the logic programming paradigm which supports and extends the basic mechanisms of the relational data model.

Indeed, both classes served as the basis of several extensions, aimed at enhancing the expressive power of the relational data model. For instance, the set of queries expressed by the relational algebra is strictly included in that of the *fixpoint queries* (the transitive closure is a fixpoint query which is inexpressible in the relational algebra), whereas it is well known that every fixpoint query can be expressed in *FO* extended with an inflationary fixpoint operator, or equivalently in Datalog extended with inflationary negation. Unfortunately, the expressiveness achieved by this kind of deterministic extensions of logic database languages is not satisfactory—e.g., the *parity* query is not a fixpoint query [6]. As a matter of fact, no known deterministic logic language expresses *exactly* the deterministic queries computable in polynomial time.

¿From a pragmatical viewpoint, a clear need for non-determinism is also emerging from applications. The *all-answers* paradigm for query execution exacerbates the need for special constructs



This work is aimed at bridging the existing gap between the two classes of proposals, by presenting an expressiveness characterization of Datalog augmented with one of the choice mechanisms, namely the *dynamic choice* construct introduced in [13]. This study is conducted both pragmatically, on the basis of examples, and formally, on the basis of known expressiveness results. In particular, we show how the dynamic choice construct is a powerful means for controlling the fixpoint computation, in order to express relevant problems such as computing the complement of a relation, or computing an arbitrary ordering of a relation.

Finally, in the main result of this paper, we show that Datalog with dynamic choice expresses exactly the non deterministic time-polynomial queries, a complexity class known as *NDB-PTIME*. The result is achieved by showing how the dynamic choice allows us to express the control needed to execute N_Datalog$\neg$ programs over ordered domains—a language which is known to capture *NDB-PTIME*. The relevance of this result is clear: Datalog with the dynamic choice has the same (high) expressiveness of languages which:

- are considerably more complex—Datalog with dynamic choice is negation-less,
- are lacking a declarative semantics—Datalog with dynamic choice is sound (although not complete) w.r.t. stable model semantics,
- are hard to be efficiently implemented—Datalog with dynamic choice is the kernel of $\mathcal{LDL}$.

As a conclusion, a simple, declarative characterization of *NDB-PTIME* is achieved by means of the dynamic choice extension of pure Datalog: such a language, although remarkably simple, is then capable of expressing all non deterministic time-polynomial queries and, therefore, all *deterministic* ones.

The plan of the paper follows. In Section 2 a survey of the main proposals of non deterministic extensions of logic database languages is provided. Particular emphasis is placed on Datalog extended with the dynamic choice construct. Section 3 and 4 show how to compute negation and ordering using the dynamic choice. Section 5 is devoted to illustrating the emulation of N_Datalog$\neg$, a language which embodies a form of nondeterminism typical of rule-based systems. Section 6 presents the main result, namely that Datalog with dynamic choice captures the complexity class *NDB-PTIME*; we then draw some conclusions, and briefly illustrate future research directions.

Close connections exist between the fixpoint FO extensions and the Datalog extensions [5]: Datalog$\neg$ expresses exactly the fixpoint queries, i.e. it is equivalent to FO+IFP. This implies that Datalog$\neg$ is strictly more expressive than Datalog with stratified negation, as the latter is known to be strictly included in FO+IFP.

Finally, the complexity measures are functions of the size of the input database. For Turing Machine complexity class C there is a corresponding complexity class of (non-deterministic) queries *(N)DB-C*. In particular, the class of (non-deterministic) database queries that can be computed by a (non-deterministic) Turing Machine in polynomial time is denoted by *(N)DB-PTIME*. No known deterministic language expresses all and only the queries in *DB-PTIME*.

2 Non-deterministic extensions of logic database languages

In this section, several mechanisms for dealing with non-determinism in logic database languages are briefly surveyed. In particular, we present a non-deterministic construct for the fixpoint extensions of FO, a non-deterministic operational semantics for Datalog$\neg$ (*à la production systems*), and a non-deterministic mechanism for pure Datalog. The first two classes of proposals are due to Abiteboul and Vianu [4, 5], whereas the third class of proposals is due to Krishnamurthy and Naqvi [18] and Giannotti, Pedreschi, Saccà and Zaniolo [20, 13].

2.1 The witness operator

A non-deterministic extension of FO is achieved by introducing the so-called *witness* operator [4, 5]. Informally, given a formula (query) $\Phi(X)$, the witness operator W_X applied to $\Phi(X)$ chooses an arbitrary X that makes Φ true. The extension of the inflationary fixpoint logic FO+IFP with the witness operator is denoted by FO+IFP+W.

Let us define more precisely the semantics of W. Notice that, in presence of non-determinism, we have a *set* of possible interpretations for a given formula in FO+IFP+W, or equivalently, a set of possible sets of answers to a given query. Consider a formula $W_X(\Phi(X,Y))$, where Y is the vector of variables other than X that occur free in Φ. Then I is an interpretation of $W_X(\Phi(X,Y))$ iff, for some interpretation J of $\Phi(X,Y)$ such that $I \subseteq J$:

- for each Y such that $\langle X,Y\rangle \in J$ for some X, there is a *unique* X_Y such that $\langle X_Y,Y\rangle \in I$.

Intuitively, one "witness" X_Y is arbitrarily chosen for each Y satisfying $\exists X.\Phi(X,Y)$. Alternatively, the meaning of W can be also described in terms of functional dependencies: the interpretation I is a maximal subset of J satisfying the functional dependency $Y \rightarrow X$.

Example 2.1 Consider a binary relation E such that $E(P,S)$ represents the fact that professor P is an eligible advisor of student S. Then the formula $W_P(E(P,S))$ realizes the non-deterministic query of assigning exactly one advisor to each student. □

It should be noted that the witness operator is added to FO independently from the fixpoint operator. Accordingly, the fixpoint computation and the non-deterministic choices do not interfere, in the sense the non-deterministic choices of the witnesses are performed w.r.t. the current fixpoint approximation, without memory of the choices that were previously operated. In other words, the witness operator performs choices *locally* to a given step of the fixpoint computation.

¿From the viewpoint of the expressive power, N_Datalog¬ is strictly included in *NDB-PTIME*. In fact, it is possible to show that such a language cannot express the query $P - \pi_1(Q)$, where P is a unary relation and Q a binary one. Thus, it is needed to extend N_Datalog¬ in order to capture all the queries in *NDB-PTIME*. Two possible approaches of remedying this problem are the following. One is allowing universal quantification in clause bodies: the resulting language is denoted N_Datalog¬∀. The second is violating the *data independence* principle, and allowing the use of *ordered* databases. In both cases we obtain languages that capture *NDB-PTIME*, and that are therefore equivalent to *FO+IFP+W*. This result is due to Abiteboul and Vianu [5].

Theorem 2.4 *A query is in NDB-PTIME iff it is expressed in N_Datalog¬∀ or, equivalently, in N_Datalog¬ over ordered databases.* □

An analogous result of the same authors shows that N_Datalog¬∗, i.e. N_Datalog¬ augmented with negation in rule heads (interpreted as deletion of facts), expresses exactly the queries in *NDB-PSPACE*.

2.3 The family of choice operators

The proposals discussed in the previous sections 2.1 and 2.2 suffer from the lack of a declarative, model-theoretic semantics, which seriously compromises their logic connotation. Another approach was started by Krishnamurthy and Naqvi [18], and later refined by Saccà and Zaniolo [20] and Giannotti, Pedreschi, Saccà and Zaniolo [13]. The proposals described in this section are based on a non deterministic *choice* construct for Datalog, which, in all cases, was designed on the basis of a declarative semantics—*choice models* in [18], and *stable models* in [20, 13]. Moreover, the choice construct can be efficiently implemented, and it is actually adopted in the logic database language $\mathcal{LDL}$[19, 7]. On the other hand, an expressiveness characterization for these proposals is lacking, which allows to compare the choice construct with the previously discussed proposals. The rest of this section surveys the original proposal and two refinements, which improve from several viewpoints.

2.3.1 Static choice

The choice construct was first proposed by Krishnamurthy and Naqvi in [18]. According to their proposal, special goals, of the form $choice((X),(Y))$, are allowed in Datalog rules to denote the functional dependency (FD) $X \rightarrow Y$. The meaning of such programs is defined by its *choice models*, as discussed next.

Example 2.5 Consider the following Datalog program with choice.

$a\_st(St, Crs) \leftarrow takes(St, Crs), choice((Crs),(St))$.
$takes(andy, engl)$.
$takes(ann, math)$.
$takes(mark, engl)$.
$takes(mark, math)$.

The choice goal in the first rule specifies that the *a_st* predicate symbol must associate exactly one student to each course. Thus the functional dependency $Crs \rightarrow St$ holds in the (choice model defining the) answer. Thus the above program has the following four choice models:

The qualification *static* for this choice operator stems from the observation that the choice is operated once and forall, after a preliminary fixpoint computation. Because of its static nature, this form of choice cannot be safely used within recursive rules. As observed in [13], the choice models semantics fails when mixed with recursion, in the sense that the delivered results do not comply with any declarative reading. Moreover, the procedure for computing choice models is extremely inefficient, as operating the choices only after a general saturation phase is wasteful—a more efficient procedure should instead operate choices as soon as possible, in order to reduce the amount of work for future saturations. Finally, due to the impossibility of being adopted within recursion, the static choice has a limited expressive power. To remedy these drawbacks, some refinements of the static choice have been proposed, which are discussed next.

2.3.2 Model-theoretical choice

An alternative approach to define a declarative semantics for the choice construct was proposed by Saccà and Zaniolo [20]. According to this proposal, programs with choice are transformed into programs with negation which exhibit a multiplicity of stable models. [1] Each stable model corresponds to an alternative set of answers for the original program. Following [20], therefore, given a choice program P, we introduce the *stable version* of P, denoted by $SV(P)$, defined as the program with negation obtained from P by the following two transformation steps:

1. Consider a choice rule of P, say

$$r : A \leftarrow B(Z), choice((X),(Y)).$$

where $B(Z)$ denotes the conjunction of all the non-choice goals of r, and Z is the vector of variables occurring in the body of r, and replace the body of r with the atom $chosen(Z)$:

$$r' : A \leftarrow chosen(Z).$$

2. add the new rule:

$$chosen(Z) \leftarrow B(Z), \neg diffChoice(Z).$$

3. add the new rule:

$$diffChoice(Z) \leftarrow chosen(Z'), Y \neq Y'.$$

where Z' is a list of variables obtained from Z by replacing variable Y by the fresh variable Y'.

The transformation directly generalizes to FD involving vectors of variables, and to multiple choice goals. When the given program P is such that none of its choice rules is recursive, then P and its stable version are semantically equivalent in the sense that the set of choice models of P coincides with the set of stable models of $SV(P)$ on common predicate symbols [20].

Example 2.6 The following is the stable version of Example 3.

[1] Stable models semantics is a concept originating from autoepistemic logic, which was applied to the study of negation in Horn clause languages by Gelfond and Lifschitz [12].

and **O** the set of the remaining (original) rules in $SV(P)$. Then, the DCF procedure operates as follows:

1. find the fixpoint of the **O** part;
2. while there exists an enabled ground instance r of a *chosen* rule in **C**, repeat:
   (a) execute r;
   (b) execute all rules in **D** enabled by r;
3. repeat steps 1 and 2 until no rule is enabled.

Notice that we used the term "execute" to mean the ordinary bottom-up computation mechanism of asserting the head of a rule whenever its body is true. The idea underlying the DCF procedure can be explained as follows. There are two modes of operation: a saturation mode and a choice mode. In the saturation mode, the consequences of the original rules are computed by an ordinary fixpoint mechanism. When nothing more can be deduced, the procedure switches to the choice mode. In the choice mode, a *chosen* rule together with the associate $diffChoice$ rules are executed, until no more choices can be made. Then the procedure switches to the saturation mode again, and the process continues until a fixpoint is reached. Notice that the execution the $diffchoice$ rules shrinks the set of enabled *choice* rules.

In other words, when DCF is in the choice mode, all the choices that are compatible with the functional dependency are operated, before DCF switches to the saturation mode again.

The DCF procedure is correct with respect to the stable model semantics, in the sense that the result of DCF is a stable model of the program $SV(P)$. However, DCF cannot compute an arbitrary stable model, but only some *preferred* ones, due to the particular policy for operating choices. Therefore, dynamic choice is sound, although not complete, w.r.t. stable model semantics [13]. ¿From now on, we shall use the term *choice model* (of a choice program) to denote one of the possible outcomes of the DCF procedure, since we restrict our attention to the dynamic choice construct.

2.3.4 A fixpoint characterization of dynamic choice

An alternative, more declarative definition of the DCF procedure can be given in terms of a non-deterministic immediate consequence operator Ψ_P associated to a choice program P. This will provide the basis of a fixpoint semantics for the dynamic choice, which will be convenient when discussing the expressive power of the dynamic choice in the next sections.

To this purpose, we adopt the transformation of a choice program P, defined by Krishnamurthy and Naqvi [18], which replaces the choice goals from the rules with new predicate symbols. A choice rule R from P:

$$R\colon\ H \ \leftarrow\ \mathbf{B}, \mathbf{C}$$

where **C** are the choice goals and **B** the other goals, is transformed into the two rules:

$$\begin{aligned} H &\leftarrow \mathbf{B}, chosen_R(Y) \\ chosen_R(Y) &\leftarrow \mathbf{B} \end{aligned}$$

$$I \in \Psi_P \uparrow n,\ J \in \Psi_P(I) \text{ implies } I \subseteq J.$$

The proof is by induction on n. In the case $n = 0$ the proof is trivial. In the induction case, the induction hypothesis is the following, for all interpretations I, J:

$$I \in \Psi_P \uparrow (n-1),\ J \in \Psi_P(I) \text{ implies } I \subseteq J.$$

Our aim is to prove, for any interpretation K:

$$K \in \Psi_P(J) \text{ implies } J \subseteq K.$$

By Def. 2.7, $J = M_I \cup I_1$, where $M_I = T_{P_{original}} \uparrow \omega(I)$, and $I_1 \subseteq T_{P_{choice}}(M_I)$.

Again, by Def. 2.7, $K = M_J \cup J_1$, where $M_J = T_{P_{original}} \uparrow \omega(J)$, and $J_1 \subseteq T_{P_{choice}}(M_J)$.

By the induction hypothesis, we have $I \subseteq J$, which implies $T_{P_{original}} \uparrow \omega(I) \subseteq T_{P_{original}} \uparrow \omega(J)$ by the monotonicity of $T_{P_{original}}$. Hence, we obtain:

$$M_I \subseteq M_J. \tag{3}$$

Moreover, from (3) and the monotonicity of $T_{P_{choice}}$ we obtain:

$$T_{P_{choice}}(M_I) \subseteq T_{P_{choice}}(M_J). \tag{4}$$

By Def. 2.7, I_1 and J_1 are maximal subsets of $T_{P_{choice}}(M_I)$ and $T_{P_{choice}}(M_J)$, respectively, which satisfy the FD's. Hence, we obtain:

$$I_1 \subseteq J_1. \tag{5}$$

from (4). Finally, we obtain the thesis from (3) and (5). □

The above proposition ensures us that $\Psi_P \uparrow \omega$ is the limit of the powers of Ψ_P, and justifies the fact that the dynamic choice fixpoint is inflationary.

Finally, the following result relates the limit of the Ψ_P operator with the output of the DCF procedure. The proof is lengthy and uninteresting, and therefore omitted.

Proposition 2.9 *Let P be a choice program. Then M is a choice model of P iff $M \in \Psi_P \uparrow \omega$.* □

The main interest for the dynamic choice construct lies in the fact that it is highly expressive—it allows to compute efficiently some relevant queries, such as negation, and ordering. These and other issues related to the expressive power of the dynamic choice are addressed in the rest of this paper.

If part. We calculate:

$$
\begin{array}{ll}
 & \models U(x) \wedge \neg P(x) \\
\Rightarrow & \quad \{ \text{By rules: } R_2, R_3, R_4, R_5 \text{ of Def. 3.1} \} \\
 & M_{NOT} \not\models TAG\_P(x,0) \wedge M_{NOT} \models TAG\_P(x,1) \\
\Rightarrow & \quad \{ \text{By rule: } R_2 \text{ of Def. 3.1} \} \\
 & M_{NOT} \models COMP\_P(x,1) \\
\Rightarrow & \quad \{ \text{By rule: } R_1 \text{ of Def. 3.1} \} \\
 & M_{NOT} \models NOT\_P(x,1).
\end{array}
$$

Only-if part. If $M_{NOT} \models NOT\_P(x)$, we have that $M_{NOT} \models chosen_{R_2}(x,1)$, since the rules R_1 and R_2 are needed to derive $NOT\_P(x)$. Assume, by contradiction, that $\models P(x)$. This implies that $M_{NOT} \models chosen_{R_2}(x,0)$, since rule R_4 derives $TAG\_P(x,0)$. Thus, $M_{NOT} \models chosen_{R_2}(x,0) \wedge chosen_{R_2}(x,1)$, which violates the FD's. □

Essentially, this example shows how the dynamic choice offers a flexible mechanism for handling the control needed to emulate the difference between two relations. It is shown in [9] that the above program can be refined in order to realize more powerful forms of negation, such as stratified and inflationary negation. This goal is achieved by suitably emulating the extra control needed to handle program strata and fixpoint approximations, respectively.

4 Ordering with the choice operator

It has been pointed out in the literature that a tight connection exists between non-determinism and ordered databases [16, 5]. On one hand, consider the case that a query Q relies on the ordering in which elements are stored in the database: when abstracted at the conceptual level, where physical details are unrelevant, Q exhibits a non-deterministic behavior. On the other hand, it is often possible to emulate ordering using non-deterministic mechanisms.

The following choice program $ORD[U]$ exploits the dynamic choice to compute an arbitrary ordering of the elements of an EDB relation U.

Definition 4.1 The choice program $ORD[U]$ consists of the following rules:

$$
\begin{array}{lll}
R_1: & SUCC(min, Y) & \leftarrow \quad U(Y), choice((), (Y)). \\
R_2: & SUCC(X, Y) & \leftarrow \quad SUCC(\_, X), U(Y), SUCC(min, Z), \\
 & & \qquad X \neq Y,\ Y \neq Z,\ choice((X), (Y)), choice((Y), (X)).
\end{array}
$$

where min is a new constant, which does not occur in the EDB. □

According to the specified operational semantics of the dynamic choice, we obtain a set of answers where the extension of relation $SUCC$ is a total, strict ordering over the input relation U. The first clause of program $ORD[U]$ starts the computation, by selecting an arbitrary element from U as the successor of min, i.e., as the actual minimum element of U. The second clause selects from U the successor y of an element x which has been already placed in order. The constraints in the body of the second clause enforce acyclity. In particular:

□

This application brings further evidence to the effectiveness of the dynamic choice as a control mechanism. It also suggests that the dynamic choice is highly expressive, as languages over ordered domains are known to be strictly more expressive than languages over unordered domains [16]. Indeed, the fact that dynamic choice can express ordering is essential in the proof of the main result of this paper.

5 Emulating N_Datalog with the choice operator

The aim of this section is to present a general transformation algorithm which allows to emulate the control needed to handle the non-deterministic semantics of N_Datalog¬. Ordering over a relation ($UNIV$) of a suitable cardinality is exploited to emulate the level of the fixpoint iteration of the N_Datalog¬ computation.

Definition 5.1 Let P be a N_Datalog¬ program. Let δ be the finite set of distinct constants occurring in P, and L the cardinality of δ. Let l be the number of distinct relations of P, and l_1 be the maximal arity of the relations in P.[2] Given an array V of distinct variables $\{V_1, \ldots, V_{l*l_1}\}$, we assume that all variables occurring in the head of a rule in P are renamed using variables from this set.

P' is a choice program obtained from P according to the following steps:

1. Add the following facts:

$$LEVEL(min).$$
$$UNIV(a_1, \ldots, a_{l_1+1}).$$

for $a_j \in \delta, j = 1, \ldots, l_1 + 1$, together with the rules of the program $ORD[UNIV]$ as in Def. 4.1. Here, min is an array (of arity $l_1 + 1$) of new constants.

2. Add the following program defining the complement of a relation P with respect to another relation U. Such program extends that of Def. 3.1 to deal with the level of the fixpoint iteration. The notation $NOT[P,U](x,n)$ is used in the following to refer to the following program.

$$NOT\_P(X,N) \leftarrow COMP\_P(X,1,N).$$

$$COMP\_P(X,I,N) \leftarrow TAG\_P(X,I,N), choice((X,N),(I)).$$

$$TAG\_P(nil,0,N) \leftarrow LEVEL(N).$$
$$TAG\_P(X,0,N) \leftarrow P(X), LEVEL(N).$$
$$TAG\_P(X,1,N) \leftarrow U(X), COMP\_P(\_,0,N).$$

where nil is a new constant.

[2]Notice that L^{l_1+1} is an upper bound for the number of instances which are derivable from P, under the natural assumption that $L \geq l$.

Example 5.2 We illustrate the transformation of Def. 5.1 on a simple N_Datalog¬ program:

$$\begin{array}{ll} R_1: & P(X), Q(Y) \leftarrow \neg R(a), S(X), T(Y). \\ R_2: & S(X) \leftarrow T(X). \\ R_3: & T(b). \end{array}$$

The following are the relevant rules of the corresponding choice program:

$$\begin{array}{lll} LEVEL(min). & & \\ NEW(X,Y,N,1) & \leftarrow & NOT[R,UNIV\_R](a,N), S(X), T(Y), \\ & & NOT[P,UNIV\_P](X,N) \\ NEW(X,Y,N,1) & \leftarrow & NOT[R,UNIV\_R](a,N), S(X), T(Y), \\ & & NOT[Q,UNIV\_Q](Y,N) \\ NEW(X,\partial,N,2) & \leftarrow & T(X), NOT[S,UNIV\_S](X,N) \\ NEW(\partial,\partial,N,3) & \leftarrow & NOT[T,UNIV\_T](a,N) \\ INSTANCE(X,Y,N,I) & \leftarrow & NEW(X,Y,N,I), choice((N),(X,Y,I)). \\ P(X) & \leftarrow & INSTANCE(X,Y,N,1). \\ Q(Y) & \leftarrow & INSTANCE(X,Y,N,1). \\ S(X) & \leftarrow & INSTANCE(X,Y,N,2). \\ T(b) & \leftarrow & INSTANCE(X,Y,N,3). \\ LEVEL(N_1) & \leftarrow & INSTANCE(\_,\_,N,\_), SUCC(N,N_1). \end{array}$$

□

The rest of this section is devoted to the proof that the transformed choice program correctly simulates the original N_Datalog¬ program. The proof is based on a bisimulation relation between the two programs: a correspondence is established between a step of the computation of the N_Datalog¬ program P and a step of the dynamic choice fixpoint computation of the transformed program P'. The key concept is the notion of *counterpart*: intuitively, an interpretation M of P' is a counterpart of an interpretation I of P if M contains the facts needed to describe the computation of P which yields I, in terms of the auxiliary relations $LEVEL$, $INSTANCE$, etc., introduced in Def. 5.1. Essentially, the proof shows that the property of being a counterpart is preserved during the computation of either programs.

Remark 5.3 Consider a choice program P' obtained with the transformation of Def. 5.1. As a consequence of Prop. 4.2, in any choice model of P' the extension of relation $SUCC$ is isomorphic to an initial fragment of the natural numbers. Therefore, to simplify the notation, we are allowed to adopt numbers to denote the elements of the relations $UNIV$ and $LEVEL$. Accordingly, if $SUCC(a,b)$ holds and a is denoted by n, we denote b by $n+1$. This convention greatly simplifies the treatment of level indicators. □

Case 1.2 $J \supset I_n$.
In this case, J is an immediate successor of I_n via some instance

$$A_0(x_0), \ldots, A_m(x_m) \leftarrow P_1(y_1), \ldots, P_k(y_k), \neg Q_1(z_1), \ldots, \neg Q_h(z_h)$$

of a rule R_i from P. By the definition of successor we have:

$$\begin{aligned} I_n &\models P_1(y_1) \wedge \ldots \wedge P_k(y_k), \\ I_n &\not\models Q_1(z_1) \vee \ldots \vee Q_h(z_h), \\ I_n &\not\models A_0(x_0) \wedge \ldots \wedge A_m(x_m). \end{aligned}$$

This, together with the fact that M is a counterpart of I_n, implies:

$$\begin{aligned} M &\models P_1(y_1) \wedge \ldots \wedge P_k(y_k), \\ M &\models NOT\_Q_1(z_1, n) \wedge \ldots \wedge NOT\_Q_h(z_h, n), \\ M &\models NOT\_A_0(x_0, n) \vee \ldots \vee NOT\_A_m(x_m, n). \end{aligned}$$

By Def. 5.1(3), we get:

$$M \models NEW(u, n, i)$$

By Def. 2.7 and Def. 5.1(4), there exists $N_1 \in \Psi_{P'}(M)$ such that:

$$N_1 \models INSTANCE(v, n, i) \tag{6}$$

corresponding to the choice represented by $chosen(n, v, i)$. The FD constraint guarantees that $N_1 \not\models INSTANCE(v', n, i')$ for any $v' \neq v$ or $i' \neq i$.

By 6 and Def. 5.1(4), we get:

$$N_1 \models LEVEL(n+1). \tag{7}$$

By 6 and Def. 5.1(5), we get:

$$N_1 \models A_0(x_0) \wedge \ldots \wedge A_m(x_m)$$

which implies:

$$N_1 \models Q(x) \quad \text{iff} \quad J \models Q(x) \tag{8}$$

for any relation Q of P, since $J = I_n \cup \{A_0(x_0), \ldots, A_m(x_m)\}$.

By Prop. 3.2 and Def. 5.1(2), there exists $N \in \Psi_{P'} \uparrow 2(N_1)$ such that:

$$N \models NOT\_Q(x, n+1) \quad \text{iff} \quad N_1 \not\models Q(x)$$

which implies, by 8:

$$N \models NOT\_Q(x, n+1) \quad \text{iff} \quad J \not\models Q(x). \tag{9}$$

By 8, 9 and Def. 5.1(3) we obtain:

$$N \models NEW(v, n+1, i) \tag{10}$$

- N_Datalog$\neg$ over ordered domains expresses *NDB-PTIME* (Theorem 2.4).

The *if* part follows from the observation that Datalog with dynamic choice is an inflationary language, as operated choices are never retracted. □

Theorem 6.1 defines precisely the expressive power of Datalog augmented with the dynamic choice construct. As a consequence, we obtain that such a language embodies a simple, declarative and efficiently implementable characterization of *NDB-PTIME*. The previous results, discussed in Section 2, adopted a combination of three mechanisms, namely fixpoint, non determinism and negation, in order to compute all (and only) the time-polynomial queries. Theorem 6.1 improves on these results, in that it implies that a suitable combination of only *two* mechanisms, fixpoint and non determinism, is enough to the purpose of computing *NDB-PTIME*.

¿From a more pragmatical viewpoint, these results indicate that dynamic choice is a flexible mechanism for explicitly handling the control in the fixpoint computation. A natural parallel here is with the *cut* control mechanism of Prolog, which is however much more difficult to be explained in declarative terms [14]. Also, it is natural to ask ourselves whether the dynamic choice provides us with the basis for constructing *bottom-up meta-interpreters*, capable of turning logic database programs into efficient systems by exploiting a customized computation strategy. Another open problem is whether it is realistic to implement negation and ordering by choice in a real language.

Finally, we mention another direction for future work. Abiteboul and Vianu showed that certain non deterministic languages augmented with the extra possibility of performing *updates* are capable of expressing *NDB-PSPACE*, i.e., the non deterministic space-polynomial queries [5]. We conjecture that a similar result holds when augmenting Datalog with dynamic choice and an update construct, such as that of $\mathcal{LDL}$.

Acknowledgments

Thanks are owing to Victor Vianu, Luigi Palopoli, Carlo Zaniolo and Mimmo Saccà for their useful suggestions on the subject of this paper. In particular, we owe the ordering example to L. Palopoli. This paper has been revised partly during the stay of two of the authors, Giannotti and Pedreschi, at CWI, Amsterdam, The Netherlands.

References

[1] K. R. Apt. Introduction to Logic Programming. In: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, vol B* (Elsevier, Amsterdam, 1990) 493-574.

[2] S. Abiteboul, E. Simon and V. Vianu. Non-Deterministic Language to Express Deterministic Transformation. In: *Proceedings of ACM Symposium on Principles of Database Systems PODS 90*, (ACM Press, 1990) 218-229.

[3] S. Abiteboul and V. Vianu. Procedural Languages for Database Queries and Updates. *Journal of Computer and System Science* 41 (2) (1990).

[20] D. Saccà and C. Zaniolo. Stable Models and Non-Determinism in Logic Programs with Negation. *Proceedings of ACM Symposium on Principles of Database Systems PODS 90*, (ACM Press, 1990) 205-217.

[21] J.D. Ullman. *Principles of Databases and Knowledge Base System. Volume I and II* (Computer Science Press, Rockville, 1988).

[22] C. Zaniolo. Object Identity and Inheritance in Deductive Databases: an Evolutionary Approach. In: *Proc. Deductive and Object-oriented Databases, First Int. Conf., DOOD'89* (Springer, Berlin, 1989)