Fundamental properties of deterministic and nondeterministic extensions of Datalog*

Serge Abiteboul and Eric Simon
I.N.R.I.A.
BP. 105, 78153 Le Chesnay

December 30, 1993

Abstract

Fundamental properties of deterministic and nondeterministic extensions of Datalog from [AV88] are studied. The extensions involve the use of negative literals both in bodies and heads of rules. Negative literals in heads are interpreted as deletions. A deterministic semantics is obtained by firing in parallel all applicable rules. The nondeterministic semantics results from firing (nondeterministically) one rule at a time. In the nondeterministic case, programs do not describe functions but relations between database states. In both cases, the result is an increase in expressive power over Datalog. The price for it is that programs do not always terminate. We study when a program (i) is such that on a given input, all its successful computations reach a unique fixpoint, (ii) yields at least one output on every input and (iii) has only loop-free computations. We also show how to simulate programs containing loops by loop-free programs.

*Work supported by the Projet de Recherche Coordonnée BD3. Email: abitebou@inria.inria.fr, simon@madonna.inria.fr

1 Introduction

The deductive database area is primarily concerned with the study of the logic programming paradigm as a way of querying a database. The *Datalog* query language (a pure Horn clause language) is a toy representative of logic-based query languages. A lot of effort has been devoted to its optimization [BR86]. Recently, many proposals emerged to develop extensions of Datalog with increased expressive power, providing forms of nonmonotonic reasoning (see, for instance [Kan88, CH80, Apt87, BH86, GS88]). The focus of the present paper is the study of extensions of Datalog proposed in [AV88, AV89]. These extensions form the basis of implementation efforts [dMS88] for so-called production-rule systems. The price to pay for the increased power is that nice properties of Datalog are lost, such as the existence of a least fixpoint, and the guarantee of program termination. In the present paper, we consider some of these fundamental properties and study under which restrictions such desirable properties continue to hold. Such studies are crucial if one hopes to offer database interfaces based on these more powerful rule-based languages.

We consider two main extensions of Datalog. The first extension (Datalog$^{\neg\star}$) allows negations in both the bodies and heads of rules, (the $\neg$ indicates that negations are allowed in bodies and the $\star$ that they are allowed in heads). Negations in heads of rules are interpreted as deletions. This allows invalidating a previously asserted fact, which is a key aspect of database updates. A second extension allows multiple literals in heads of rules that are either positive or negative. We call this extension N-Datalog$^{\neg\star}$. Different semantics are assigned to these languages, which entail two families of deterministic and nondeterministic languages. The choice between deterministic and nondeterministic semantics results from the choice to consider one possible application of a rule at one time, or to apply all possible instantiations of the rules in parallel. The deterministic semantics is assigned to Datalog$^{\neg\star}$ programs. The nondeterministic semantics is assigned to N-Datalog$^{\neg\star}$ programs ("N" stands for nondeterministic.) These two languages can be viewed both as query languages and as update languages. We will therefore distinguish between *query programs* that do not modify the input relations and *(arbitrary) programs* that may modify them.

The expressive power of these languages is studied in [AV88] and bridges with procedural languages of [AV87] are exhibited. Connections between these languages and fixpoint extensions of first-order logic are investigated in [AV89].

We consider here three important properties of the Datalog extensions:

1. The first property, called *totalness*, holds when a program describes a "total" relation between database instances. By this, we mean that the program always admits at least one output on every input.
2. The second property, called *loop-freeness*, guarantees that on every input, the program never enters an infinite loop. In other words, for each input, each

computation terminates.

3. The last property, called *functionality*, expresses that on a given input, all successful computations reach a unique fixpoint (This can be viewed as a Church-Rosser property.) There may be computations that go into infinite loops.

Clearly, these three properties are important for implementation purposes (see [dMS88]). Furthermore, the study of these properties brings new insights into the nondeterministic semantics and the use of negative literals in heads.

We systematically study each one of the three properties for each Datalog extension and for sublanguages. Surprisingly, queries and arbitrary programs behave differently with respect to them. Less surprisingly, we show that, in most cases, when a property does not hold in general for a given (sub)language, the property is undecidable. Besides this main theme, the paper provides the following related contributions:

1. It is important to know when the deterministic and nondeterministic semantics coincide to be able (for efficiency reason) to implement the nondeterministic semantics using the deterministic one. We study this issue and doing so, answer an open problem of [SdM88].

2. Although loops are inherently present as soon as "deletions" are introduced, there is a subtle way of avoiding them. We introduce the notion of "loop-free simulation" of programs with loops and prove that for both deterministic and nondeterministic programs, such simulations always exist.

The paper is organized as follows. Section 2 gives the necessary background on the Datalog extensions that are studied and introduces the properties that are considered. Section 3 is concerned with totalness and loop-freeness. Section 4 is devoted to the study of functionality. The deterministic and nondeterministic semantics are compared in Section 5. In Section 6, we consider the simulation of programs containing loops by loop-free programs. Finally, the last section is a conclusion. Figure 1 summarizes the results of Sections 3 and 4, and Figure 2 that of Section 5. (See at the end of the paper.)

2 Preliminaries

In this section, we briefly recall the languages of [AV88] that are considered in the present paper. We also introduce the three properties that are studied.

We assume that the reader is familiar with the basic concepts and terminology of relational databases [Ull88]. We also refer to [Kan88] for a survey of the field. We first review some database terminology and notation.

We assume the existence of three infinite and pairwise disjoint sets of symbols: the set of *predicates*, the set of *constants*, and the set of *variables*. With each predicate, is associated a particular integer called the *arity*. A *fact* over a predicate R of arity n is an expression of the form $R(a_1, ..., a_n)$ where each a_i is a constant. A *database schema* is a finite set of predicates. A *(database) instance* over a schema $\mathcal{S}$ is a finite set of facts over predicates in $\mathcal{S}$. Let I be a set of facts and Q a predicate in S. Then $I[Q]$ is the set of facts over Q in I.

Definition 2.1 A *literal* is an expression of the form $(\neg)Q(x_1, ..., x_m)$ where $m \geq 0$, Q is a predicate of arity m and each x_i is a variable. An *eq-literal* is an expression of the form $(\neg)x_1 = x_2$ where x_1, x_2 are variables.

The first Datalog extension is obtained by allowing negative literals in both the bodies and the heads of rules.

Definition 2.2 A *Datalog*$^{\neg\star}$ *rule* is an expression of the form

$$A \leftarrow B_1, ..., B_n$$

$(n \geq 0)$ where A and each B_i are literals.

A *Datalog*$^{\neg\star}$ *program* is a pair $< \Gamma, \mathcal{S} >$ where Γ is a finite set of *Datalog*$^{\neg\star}$ rules and $\mathcal{S}$ a set of predicates. (The meaning of $\mathcal{S}$ will be given later.) If the literals in heads of rules are all positive, the program is also a *Datalog*$^{\neg}$ *program*; and if the literals in bodies are all positive, the program is also a *Datalog** *program.* If all literals are positive, the program is a *Datalog program.*

Note that the programs that we consider do not have occurrences of constants. This is in order to study a "pure" language. Constants can be added easily without changing the framework.

These Datalog languages are further extended by allowing multiple literals in the heads of rules.

Definition 2.3 An *N-Datalog*$^{\neg\star}$ *rule* is an expression of the form

$$A_1, ..., A_k \leftarrow B_1, ..., B_n$$

$(k \geq 1, n \geq 0)$, where each A_j is a literal and each B_i is a literal or an eq-literal.

An *N-Datalog*$^{\neg\star}$ program is a pair $< \Gamma, \mathcal{S} >$ where Γ is a finite set of *N-Datalog*$^{\neg\star}$ rules and $\mathcal{S}$ a set of predicates. If the literals in heads are all positive, the program is also an *N-Datalog*$^{\neg}$ *program*; and if the literals in bodies are all positive, the program is also an *N-Datalog** *program*. If all literals are positive, the program is an *N-Datalog program*.

Intuitively, a program $< \Gamma, \mathcal{S} >$ defines a mapping from instances over $\mathcal{S}$ to instances over the predicates occurring in the program. (The predicates in $\mathcal{S}$ are called *input* predicates.) The languages in the second extension are called N-Datalog languages because they will be assigned a nondeterministic semantics. On the contrary, Datalog languages in the first extension will be assigned a deterministic semantics. Note the difference in syntax between deterministic (i.e., Datalog) and nondeterministic (i.e., N-Datalog) rules: nondeterministic rules may have several literals in the head, and may use equality in the body. From the semantics we shall describe, it will become clear that the additional features would be redundant in the deterministic case.

Definition 2.4 Let r be a Datalog$^{\neg\star}$ rule. Let I be a set of facts and r' be a ground instance of r such that (i) each literal of the body is a fact in I and (ii) each variable is valuated to some constant occurring in I. Then the ground literal of the head of r' is called an *immediate consequence* of I using r. The set of all the immediate consequences of I using a set of rules Γ is denoted $imm\_cons_\Gamma(I)$.

Intuitively, the set of immediate consequences of I using rules in Γ is obtained by firing in parallel all the rules for all possible valuations of rules in Γ that are applicable in I.

Deterministic semantics: Let Γ be a set of Datalog$^{\neg\star}$ rules. Γ also denotes a mapping over sets of facts defined by: for each I, $(I,\ J)$ is in Γ where J consists of the facts A such that:

- A is in $I \cup imm\_cons_\Gamma(I)$ and $\neg A$ is not in $imm\_cons_\Gamma(I)$ or
- A is in I and A, $\neg A$ are both in $imm\_cons_\Gamma(I)$.

If the sequence $\Gamma^1(I)$, $\Gamma^2(I)$,... has a limit, it is denoted $\Gamma^\infty(I)$.

Note that the deterministic semantics of a program can be viewed as a function or alternatively as a relation among database instances (i.e., the graph of the function).

The language Datalog$^{\neg}$ with the above semantics has been independently introduced in [KP87, AV88].

To introduce the nondeterministic semantics, we define a different notion of immediate consequences of a set of facts using a rule. Let r be an N-Datalog$^{\neg\star}$ rule. Let I be a set of facts and r' be a ground instance of r such that (i) each literal of the body of r' is a fact in I and each eq-literal of the body of r' holds, (ii) the head of r' is consistent and (iii) each variable is valuated to some constant occurring in I. Then the set of literals in the head of r' is called an *immediate consequence* of I using r'.

By condition (ii) above, a ground instance of a rule is not considered if it contains a ground literal A and its negation.

Nondeterministic semantics: Let Γ be an N-Datalog$^{\neg\star}$ program. Γ defines a relation over sets of facts, denoted Γ_n, as follows: for each I, (I,J) is in Γ_n if for some immediate consequence $A_1,...,A_p,\neg B_1,...,\neg B_q$ of I using some instantiation of a rule in Γ:

$$J = (I \cup \{A_1,...,A_p\}) - \{B_1,...,B_q\}.$$

Some pair (I,J) is in Γ_n^∞ iff there exists a sequence $I_0 = I,...,I_n = J$ such that (i) for each i, $(I_i, I_{i+1}) \in \Gamma_n$ and (ii) there is no $J' \neq J$ with $(J,J') \in \Gamma_n$.

We next introduce the main properties that are studied in the paper. We present them in the case of nondeterministic programs and then consider the deterministic case.

Let $<\Gamma,\mathcal{S}>$ be an N-Datalog$^{\neg\star}$ program.

totalness: If for each input I over $\mathcal{S}$, there is some J such that (I,J) is in Γ_n^∞, we shall say that $<\Gamma,\mathcal{S}>$ is *total.*

loop-freeness: If there is no infinite sequence $\{I_n\}$, $n \geq 0$, such that (i) I_0 is an instance over $\mathcal{S}$, (ii) for each i, $(I_i, I_{i+1}) \in \Gamma_n$ and (iii) for each i, $I_i \neq I_{i+1}$, we shall say that $<\Gamma,\mathcal{S}>$ is *loop-free.*

functionality: Let Q be a predicate. If

$$\{\ (I, J[Q]) \mid I \text{ over } \mathcal{S},\ (I,J) \in \Gamma_n^\infty\ \}$$

is the graph of a (partial) function, we shall say that $<\Gamma,\mathcal{S}>$ is *functional for Q*. If $<\Gamma,\mathcal{S}>$ is functional for each predicate, we say that $<\Gamma,\mathcal{S}>$ is *functional.*

These notions are defined similarly for Datalog$^{\neg\star}$ programs. Note that by definition, Datalog$^{\neg\star}$ programs are functional. Also, loop-freedom implies totalness for N-Datalog$^{\neg\star}$. For Datalog$^{\neg\star}$, totalness is equivalent to loop-freedom.

Query programs
Programs are often used to query the database. In the context of queries, it is traditional to distinguish between:

- the *extensional predicates* (EDB) that occur only in the bodies of rules, and
- the *intensional predicates* (IDB) that occur in heads of rules (and possibly also in bodies).

The intuition is that the input is an instance over the EDB predicates and the program does not modify the input. In that spirit, a program $< \Gamma, \mathcal{S} >$ is a *query* if $\mathcal{S}$ is the set of predicates which do not occur in heads of rules (i.e., the EDB predicates). We use Γ as a shorthand for a query $< \Gamma, \mathcal{S} >$ since $\mathcal{S}$ is determined by Γ.

3 Totalness and loop-freedom

In this section, we study totalness and loop-freedom. We identify languages where programs are total and loop-free. For other languages, we prove that these properties cannot be guaranteed, and that one cannot decide in general whether given programs satisfy them.

3.1 Basic properties

Theorem 3.2 below states that certain classes of programs are always total and loop-free. To show Theorem 3.2 (ii), we use a technical lemma which shows that Datalog* queries are essentially *inflationary*. By this, we mean that a computation of a Datalog* query on an input I, consists in deriving new facts without ever invalidating previously asserted facts. More formally,

Lemma 3.1 Let Γ be a Datalog* query and I an instance over the EDB predicates of Γ. Then for each i, $\Gamma^i(I) \subseteq \Gamma^{i+1}(I)$.

Proof: The proof is by induction. Since the EDB predicates are not modified, $I \subseteq \Gamma^1(I)$. Suppose that for some i, $\Gamma^i(I) \subseteq \Gamma^{i+1}(I)$. Let $Q(\vec{a})$ be in $\Gamma^{i+1}(I)$. Two cases arise:

1. Q is an EDB predicate, so $Q(\vec{a})$ is in $\Gamma^{i+2}(I)$.
2. Q is an IDB predicate. Since $Q(\vec{a})$ is in $\Gamma^{i+1}(I)$, $Q(\vec{a})$ is an immediate consequence of $\Gamma^j(I)$ using Γ for some j in [0,i]. By the induction hypothesis and

since there are no negative literals in rule bodies of Γ, $Q(\vec{a})$ is an immediate consequence of $\Gamma^{i+1}(I)$ using Γ. Since $Q(\vec{a})$ is an immediate consequence of $\Gamma^{i+1}(I)$ using Γ and is in $\Gamma^{i+1}(I)$, $Q(\vec{a})$ is in $\Gamma^{i+2}(I)$ by definition of the deterministic semantics.

Thus, $\Gamma^{i+1}(I) \subseteq \Gamma^{i+2}(I)$. $\Box$

Remark: The previous result shows that Datalog* queries are inflationary. One would be then tempted to believe that given such a query Γ, an equivalent query is obtained by erasing all rules with negative literals in heads. In fact, this needs not be the case. Rules with negative heads, although not used for deleting existing tuples, may be used to inhibit the derivation of new tuples. This results in an increased power over Datalog programs. Indeed, one obtains exactly the power of Datalog$^\neg$ queries.

Theorem 3.2 (i) Datalog$^\neg$ and N-Datalog$^\neg$ programs are total and loop-free. (ii) Datalog* queries are total and loop-free.

Proof: The proof of (i) is straightforward. To see (ii), we use Lemma 3.1. By this lemma, Datalog* queries are total and loop-free by finiteness. $\Box$

The next results state that totalness (and therefore loop-freedom) cannot be guaranteed in the other cases.

Proposition 3.3

1. N-Datalog* queries are not always total.
2. Datalog$^{\neg\star}$ queries are not always total.
3. Datalog* programs are not always total.

Proof:

To see (1), consider the N-Datalog* query consisting of the single rule:

$$P(x,y), \neg P(y,x) \leftarrow Q(y,x)$$

with input $\{Q(0,1), Q(1,0)\}$.

To see (2), consider the behavior of the Datalog$^{\neg\star}$ query[1]:

[1]We use Datalog rules with multiple literal heads as "macros" with the obvious semantics.

$$
\begin{aligned}
&P(x,y) \leftarrow \neg stepone, Q_0(x), Q_1(y),\\
&stepone \leftarrow,\\
&\neg P(x,y), P(y,x) \leftarrow P(x,y)
\end{aligned}
$$

on input $I = \{Q_0(0), Q_1(1)\}$. Observe that the simpler query

$$
\begin{aligned}
&P(x,y) \leftarrow Q_0(x), Q_1(y),\\
&\neg P(x,y), P(y,x) \leftarrow P(x,y)
\end{aligned}
$$

is loop-free by Theorem 3.2. For instance, on input I, once $P(0,1)$ has been derived by the first rule, this first rule is still applicable and prevents the second rule from erasing it.

For (3), consider the Datalog* program $< \Gamma, \{P\} >$ where Γ consists of the single rule:

$$\neg P(x,y), P(y,x) \leftarrow P(x,y). \ \Box$$

In a next proposition, we relate the two properties. To prove it, we need an analog of Lemma 3.1 for N-Datalog*. By this lemma, N-Datalog* queries are (roughly speaking) inflationary. Other properties of N-Datalog* queries such as functionality will easily follow from the lemma.

Lemma 3.4 Let Γ be an N-Datalog* query and I an instance over the EDB predicates of Γ. Let $(I_0 = I, ..., I_n = J)$ be a computation of Γ on input I reaching a fixpoint J. Then (i) for each i, $I_i \subseteq I_{i+1}$ and (ii) each computation of Γ on input I terminates at J.

Proof: First consider (i). The proof is by induction. It is obvious for $i = 0$. Suppose that it holds for some i. Suppose that A is some fact in $I_{i+1} - I_{i+2}$. Two cases occur:

1. $A \notin J$. Since EDB predicates are not modified, $I \subseteq J$. Thus, since there are no negative literals in bodies, the sequence of rules which led to introducing A from I is still applicable in J. Hence J is not a fixpoint, a contradiction.
2. $A \in J$. For similar reasons as in (1), there is a sequence of rule applications leading to the deletion of A. Thus J is not a fixpoint, a contradiction.

Thus, by (1) and (2), $I_{i+1} \subseteq I_{i+2}$.

Now to see (ii), consider a computation of Γ on input I. Two cases occur:

- The computation terminates at a fixpoint J'. We first show that $J \subseteq J'$. By (1), no fact derived in a computation leading to a fixpoint is deleted. If A is a fact in J, there is a sequence of rules deriving A from I. Since $I \subseteq J'$ and there are no negative literals in bodies, the same sequence of rules can be used to derive A from J'. Since J' is a fixpoint, $A \in J'$. Thus, $J \subseteq J'$. By symmetry, $J' \subseteq J$, so $J = J'$.

- The computation $I'_0 = I, ..., I'_i, ...$ is nonterminating. By finiteness, there exists i with $I'_i \not\subseteq I'_{i+1}$. Thus some fact t has been derived and then deleted. Since there is no negation in the rules, and $I \subseteq J$, the sequence of rules deriving t is applicable from J, so t is in J, as J is a fixpoint. By the same argument, the sequence of rules deleting t is applicable from J, so J is not a fixpoint, a contradiction. □

Note in the proof the crucial use of the fact that input predicates are not modified. If this is relaxed, the result does not hold. Observe also that the previous lemma does not imply that N-Datalog* query computations never delete tuples. However, it implies that if a tuple is deleted in a step, the computation is nonterminating.

We are now ready to state:

Proposition 3.5

1. A Datalog$^{\neg\star}$ program is loop-free iff it is total.

2. An N-Datalog* query is loop-free iff it is total.

3. An N-Datalog* program may be total without being loop-free.

Proof: The proof of (1) is straightforward. (2) is a direct consequence of Lemma 3.4. To see (3), consider the N-Datalog* program $< \Gamma, \{P\} >$ with Γ consisting of:

$$\neg P(x,y), P(y,x) \leftarrow P(x,y),$$
$$\neg P(x,y) \leftarrow .$$

On input $I = \{P(0,1)\}$, an infinite loop can be found. However, the program is total since (by the second rule) the empty instance is a valid output for any input. □

3.2 Deciding totalness and loop-freedom

In this section, we show undecidability results for totalness and loop-freedom. We exhibit a decision procedure only for an important subcase.

We first consider the deterministic languages with negation in heads (i.e. Datalog* and Datalog$^{\neg\star}$) first for programs, then queries.

Theorem 3.6 (i) It is undecidable, given a Datalog* program Γ, whether Γ is loop-free (total), and (ii) it is undecidable, given a Datalog$^{\neg\star}$ query Γ whether Γ is loop-free (total).

Proof: We first prove (i). The proof is by reduction from the FD-implication problem for Datalog queries:

FD-implication for Datalog[2]: Given a Datalog query Γ, a functional dependency (FD) $R : 1 \to 2$ over some binary EDB predicate R of Γ, and an FD $S : 1 \to 2$ over some binary IDB predicate S of Γ, is it true that for each instance I over the EDB predicates,

$$I \models R : 1 \to 2 \text{ implies } \Gamma^\infty(I) \models S : 1 \to 2?$$

Fact [AH88]: The FD-implication problem for Datalog is undecidable.

Let $< \Gamma_1, R : 1 \to 2, S : 1 \to 2 >$ be an instance of the FD-implication problem for Datalog. Let $flip, flop, continue$ be new unary predicates and $\hat{R}$ a binary predicate. Consider the Datalog* program $< \Gamma, \mathcal{S} >$defined by:

- $\mathcal{S}$ consists of the EDB predicates of Γ_1 together with *flip, flop, continue* and
- Γ consists of the rules of Γ_1 and of the following rules:
  1. (a) $\hat{R}(y, y') \leftarrow R(x, y), R(x, y')$.
     (b) $\neg\hat{R}(y, y) \leftarrow .$
     (c) $\neg continue(w) \leftarrow \hat{R}(y, y')$.
  2. $\neg continue(w) \leftarrow flip(y), flop(y)$.
  3. $\neg flip(y), \neg flop(z), flip(z), flop(y) \leftarrow$
     $flip(y), flop(z), S(x, y), S(x, z), continue(w)$.

Claim : Γ is loop-free iff the FD-implication problem has a positive answer for $< \Gamma_1, R : 1 \to 2, S : 1 \to 2 >$.

Only-if part: Assume that the FD-implication problem has a negative answer for $< \Gamma_1, R : 1 \to 2, S : 1 \to 2 >$. We show that Γ is not loop-free. Let J be the fixpoint of Γ_1 on an input I such that $I \models R : 1 \to 2$, and $J \not\models S : 1 \to 2$. Let $S(a, b), S(a, b')$ be two facts in J with $b \neq b'$. (Such facts exist since $J \not\models S : 1 \to 2$.) Let I' be an instance of $\mathcal{S}$ such that:

[2] We restrict somewhat the problem by requiring that the predicates R and S are binary. The undecidability is shown in [AH88] for this restricted version of the problem.

- for each EDB predicate Q of Γ_1, $I'[Q] = I[Q]$,
- $I'(flip) = \{b\}, I'(flop) = \{b'\}$, and *continue* is nonempty.

We consider the computation of Γ on input I'. By construction, $I' \models R : 1 \to 2$, and Rule (1-b) prevents the insertion of tuples in $\widehat{R}$. Now, since *flip* and *flop* have an empty intersection, Rule 2 is never applicable, and *continue* is never emptied. Finally, Rule 3 enters an infinite loop because of the dependency violation in S.

If part: Assume that the FD-implication problem has a positive answer for $< \Gamma_1, R : 1 \to 2, S : 1 \to 2 >$. We show that Γ is loop-free. Let I' be an instance over the EDB predicates of Γ and I the projection of I' over the EDB predicates of Γ_1. The following cases arise:

- $I' \not\models R : 1 \to 2$.
  Then some tuple is entered in $\widehat{R}$ by Rule (1-a) and *continue* is emptied at the second step by Rule (1-c). After that, a fixpoint is reached when Γ_1 saturates.
- $I' \models R : 1 \to 2$.
  Then, Rule (1-b) prevents the insertion of tuples in $\widehat{R}$. Two cases arise:
  1. If *flip*, *flop* or *continue* is empty in I.
     Then a fixpoint is reached when Γ_1 saturates.
  2. If *flip* and *flop* have a nonempty intersection in I', *continue* is emptied at the first step and a fixpoint is reached when Γ_1 saturates.
  3. Otherwise, since $I \models R : 1 \to 2$, $J \models S : 1 \to 2$, so Rule 3 is never applicable. Therefore a fixpoint is also reached when Γ_1 saturates.

We next consider part (ii) of the Theorem. The proof is similar to the proof of part (i). The difficulty here is that the EDB predicates (e.g., *flip*) cannot be modified. So they are first copied in a first step into new IDB predicates. The simulation is next performed on the copies. Note that the language provides the necessary control to connect up the copy step and the simulation one. □

We now turn to the nondeterministic case. We next exhibit a decision procedure for N-Datalog* queries. To prove it, we use a reduction to Datalog$^{\neq}$ (i.e., to Datalog extended with inequalities in rule bodies) *satisfiability*. The satisfiability problem for a language L is as follows: given a query Γ in L and a predicate Q occurring in Γ, does there exist a database I over the EDB predicates of Γ such that $\Gamma^\infty(I)[Q] \neq \emptyset$. It is known that the satisfiability problem is decidable for Datalog [Shm87]. We first extend that result to Datalog$^{\neq}$. Note that the same proof also works if constants are allowed in programs.

Proposition 3.7 One can decide, given a Datalog$^{\neq}$ query Γ and an IDB predicate Q of Γ, whether Γ is satisfiable for Q.

Proof: Let $\mathcal{S}$ be a schema and n an integer. Let $I(\mathcal{S}, n)$ be the set of instances over $\mathcal{S}$ defined as follows. An instance I is in $I(\mathcal{S}, n)$ if (i) there are exactly n constants occurring in I and (ii) I is the set of all facts over predicates in $\mathcal{S}$ that can be built with these n constants.

Let Γ be a Datalog$^{\neq}$ query, n the maximum number of variables in a rule of Γ, $\mathcal{S}$ the set of EDB predicates in Γ and Q some IDB of Γ. The n constants serve the purpose of allowing the various possible inequalities in the body of a rule. We show that

(†) Γ is satisfiable for Q, iff there exists an instance $I(\mathcal{S}, n)$ such that $\Gamma^{\infty}(I)[Q] \neq \emptyset$.

For suppose that this is the case. Then it clearly suffices to choose n constants and check whether $\Gamma^{\infty}(I)[Q] \neq \emptyset$ for the maximum instance I over $\mathcal{S}$ built with these n constants.

To prove (†), we show by induction that for each k:

(‡) some fact A is derivable by Γ from some instance over $\mathcal{S}$ in k steps, iff A is derivable by Γ from an instance $I(\mathcal{S}, n)$.

Basis of the induction: obvious.

Induction: Suppose that (‡) holds for some k and that A is derivable by Γ from some instance over $\mathcal{S}$ in $k+1$ steps . Let

$$A \leftarrow B_1, ..., B_m$$

be the ground instance of a rule that is used in the $k+1$-th step. Then, each B_i is derivable in k steps. By induction, each B_i is also derivable by Γ from an instance $I(\mathcal{S}, n)$. By isomorphism, they are all derivable from the same instance I. Hence, A is derivable from an instance $I(\mathcal{S}, n)$. By induction, (‡) holds. Hence (†) is proven. $\square$

Using this proposition and Lemma 3.4, we have:

Theorem 3.8 One can decide, given an N-Datalog* query Γ, whether Γ is loop-free or total.

Proof: (sketch) As mentioned above, the proof is by reduction to the satisfiability of Datalog$^{\neq}$ queries. Let Γ be an N-Datalog* query. We use two programs Γ_1 and Γ_2 which can be viewed as computing the positive and negative parts of Γ. Γ_1 is a Datalog$^{\neq}$ program that for each input I, computes all the facts that can be derived from I using Γ. Γ_2 is a Datalog$^{\neq}$ program computing the facts that can be potentially "invalidated" by rules in Γ. Γ_2 computes on a set of new IDB predicates (say $\{\widehat{Q} \mid Q$ is an IDB predicate occurring in $\Gamma\}$) to separate the two computations. Let

$$(\dagger) A_1(\vec{x}_1), ..., A_k(\vec{x}_k), \neg B_{k+1}(\vec{x}_{k+1}), ..., \neg B_l(\vec{x}_l) \leftarrow \mathit{body}$$

be a rule in Γ. Then Γ_1 and Γ_2 contain rules simulating respectively the positive and negative fragments of $(\dagger)$:

1. A rule of Γ_1 is obtained by applying the following sequence of modifications to $(\dagger)$: for each j, $k+1 \leq j \leq l$, remove $\neg B_j(\vec{x_j})$ and for each i with $B_j = A_i$, add the inequality[3] "$\vec{x}_j \neq \vec{x}_i$" to the body of the rule.
2. A rule of Γ_2 is obtained by applying the following sequence of modifications to $(\dagger)$: for each j, $1 \leq j \leq k$, remove $A_j(\vec{x_j})$ and for each i with $A_j = B_i$, add the inequality "$\vec{x}_j \neq \vec{x}_i$" to the body of the rule. Also, replace each predicate $\neg B_k(\vec{x_k})$ in the head by $\widehat{B_k}(\vec{x_k})$. Finally, each IDB predicate C in the body of the rule is replaced by $\widehat{C}$.

Note that the inequality is needed.

By Lemma 3.4, the instance resulting from applying Γ_1 on input I can be viewed as *the* candidate for being the result of Γ on I. Now, suppose that Γ has an infinite computation, then by finiteness, some tuple will have to be first derived then deleted. This tuple will eventually be both in Q and $\widehat{Q}$. Thus, Γ is total or loop-free iff for each input I, no fact can be derived both by Γ_1 and Γ_2. Therefore consider the program Γ' consisting of Γ_1, Γ_2 and the rule:

$$\mathit{ok} \leftarrow Q(\vec{x}), \widehat{Q}(\vec{x})$$

for $\vec{x}$ a vector of distinct variables. Then, Γ' is satisfiable for *ok* iff Γ is not total iff Γ is not loop-free. $\Box$

To conclude this section, we consider the cases of N-Datalog* programs and N-Datalog$^{\neg\star}$ queries.

[3] We allow here Datalog$^{\neq}$ programs with several literals in heads of rules and with inequalities of the form $\vec{u} \neq \vec{v}$ in the bodies. These features (with the obvious semantics) can be viewed as "macros". It is straightforward to transform a rule in this extended language into a set of conventional Datalog$^{\neq}$ rules.

Theorem 3.9 (i) It is undecidable, given an N-Datalog* program Γ, whether Γ is loop-free (total), and (ii) it is undecidable, given a N-Datalog$^{\neg\star}$ query Γ, whether Γ is loop-free (total).

Proof: We first consider Part (i). The proof is again by reduction from the FD-implication problem of Datalog queries. It resembles that of Theorem 3.6.

Let $< \Gamma_1,\ R : 1 \rightarrow 2,\ S : 1 \rightarrow 2 >$ be an instance of the FD-implication problem for Datalog. Let $H, H', \widehat{R}$ be new predicates of respective arities 1,1,2. Then, consider the N-Datalog* program $< \Gamma, \mathcal{S} >$, where

- $\mathcal{S}$ consists of the set of EDB predicates of Γ_1 with the exception of R, together with H, H' and $\widehat{R}$, and
- Γ consists of the rules of Γ_1 and of the following rules:
    1. $R(x, y), \neg H(x) \leftarrow \widehat{R}(x, y), H(x)$.
    2. $H'(y), \neg H'(z) \leftarrow S(x, y), S(x, z)$.

Claim: Γ is loop-free iff the FD-implication problem has a positive answer for $< \Gamma_1,$ $R : 1 \rightarrow 2,\ S : 1 \rightarrow 2 >$.

Only if part: We show the contrapositive. Assume that the FD-implication problem has a negative answer. Then, there exists an instance I over the EDB predicates of Γ_1 such that $I \models R : 1 \rightarrow 2$ and $\Gamma_1^\infty(I) \not\models S : 1 \rightarrow 2$. Let I' be an instance over the EDB predicates of Γ such that:

(i) for each EDB predicate Q of Γ_1, with the exception of R, $I[Q] = I'[Q]$, and

(ii) $I'[\widehat{R}] = I[R]$ and $I'[H]$ is the projection on the first attribute of $I[R]$.

We consider the computation of Γ on input I'. By Rule 1, the content of R will satisfy $R : 1 \rightarrow 2$. Also, when Rule 1 saturates, the content of R is exactly $I[R]$. Now, by hypothesis, a dependency violation arises in S after computing the rules of Γ_1 up to saturation. Hence, by Rule 2, the program loops forever.

If part: Let I be an instance over the EDB predicates of Γ. By Rule 1, $I[R] \models R :$ $1 \rightarrow 2$. By hypothesis, the dependency is never violated in S. Thus, variables y and z in Rule 2 can only be valuated to the same constant, and the head of the rule is inconsistent. Thus, a fixpoint is reached when the rules of Γ_1 saturate and Γ is loop-free.

We now come to Part (ii). The proof is similar to the proof of part (i). The difficulty is that now the EDB predicates cannot be modified. So they are first copied into new IDB predicates. (More precisely, a subset of the input is nondeterministically copied first.) The simulation is next performed on the copies of the EDB predicates.

Note that the language provides the necessary control to connect up the copy and the simulation steps. □.

4 Functionality

In this section, we study the functionality property. Programs with the deterministic semantics are functional. We show that N-Datalog programs and N-Datalog* queries are also functional. In all other cases, the functional property cannot be guaranteed. Furthermore, one cannot decide whether a program is functional.

4.1 Basic properties

Theorem 4.1 (i) Datalog$^{\neg\star}$ programs, (ii) N-Datalog programs, and (iii) N-Datalog* queries are functional.

Proof: (i) is true by definition, (ii) is obvious and (iii) is by Lemma 3.4. □

We also have:

Proposition 4.2 (i) There exist N-Datalog$^\neg$ queries that are not functional; and (ii) there exist N-Datalog* programs that are not functional.

Proof: To show (i), consider the N-Datalog$^\neg$ query consisting of the single rule:

$$P(x, y) \leftarrow Q(x, y), \neg P(x, y), \neg P(y, x)$$

and input $I = \{Q(0, 1), Q(1, 0)\}$.

To see (ii), consider the N-Datalog* program $< \Gamma, \{P\} >$ where Γ consists of the single rule:

$$\neg P(x, y) \leftarrow P(x, y), P(y, x)$$

and the input $I = \{P(0, 1), P(1, 0)\}$. □

4.2 Undecidability of the functionality property

In this section, we prove that the functionality property is undecidable for N-Datalog$^\neg$ queries and N-Datalog* programs.

Theorem 4.3 It is undecidable, given an N-Datalog$^\neg$ query Γ, and a predicate R whether Γ is functional for R.

Proof: The proof is by reduction from the containment of Datalog queries:

Containment[4]: given two Datalog queries, Γ_1 and Γ_2, Γ_1 is contained by Γ_2 for Q, noted $\Gamma_1 \subseteq_Q \Gamma_2$, iff for every instance I over the EDB predicates, $\Gamma_1^\infty(I)[Q] \subseteq \Gamma_2^\infty(I)[Q]$.

Fact [Shm87]: One cannot decide given two Datalog queries Γ_1 and Γ_2 over the same EDB predicates and with one common IDB predicate Q, whether $\Gamma_1 \subseteq_Q \Gamma_2$.

Let Γ_1 and Γ_2 be two Datalog queries over the same EDB predicates and with one common IDB predicate Q. Let Γ_2' be the query obtained by marking IDB predicates in Γ_2 to distinguish them from predicates in Γ_1. Suppose Q' is the marked version of Q.

Let $steptwo, H, R$ be three new 0-ary predicates. Then consider the N-Datalog$^\neg$ query Γ consisting of the following rules:

1. $head \leftarrow body, \neg steptwo$
   for all rules: $head \leftarrow body$ in Γ_1 or Γ_2'.
2. $steptwo \leftarrow$.
3. $H \leftarrow Q(\vec{x}), \neg Q'(\vec{x}), steptwo$
   where $\vec{x}$ is a vector of distinct variables.
4. $R \leftarrow \neg H$.
5. $R \leftarrow body, \neg A, steptwo$
   for all rules: $head \leftarrow body$ in Γ_1 or Γ_2' and A is the literal in $head$.

It suffices to show:

$$(\dagger)\ \Gamma_1 \subseteq_Q \Gamma_2 \text{ iff } \Gamma \text{ is functional for } R.$$

Suppose first that the containment holds. Two cases occur in the computation of the query Γ:

- Rules 1 are applied first to saturation. Because of the containment, Rule 3 is never applied and R is derived by Rule 4.

[4] Our formulation is slightly different but equivalent to that of [Shm87].

- Rule 2 is applied before Rules 1 saturate. Then R is derived by Rule 5.

Thus R is always derived and Γ is functional in R.

Conversely, suppose that the containment does not hold. Let I be an input such that

$$\Gamma_1^\infty(I)[Q] \not\subseteq \Gamma_2^\infty(I)[Q].$$

Consider the following two computations of Γ on input I:

- Rule 4 is applied first to derive R.
- Rules 1 are applied to saturation. Next Rule 2 is applied, then H is derived using Rule 3. Rules 4 and 5 will never become applicable, so R is not derived.

Thus Γ is not functional in R. $\Box$

Remark: Note that undecidability of functionality for an IDB predicate Q does not imply undecidability of "global functionality" (i.e., functionality for all predicates).

Theorem 4.4 It is undecidable, given an N-Datalog* program $< \Gamma, \mathcal{S} >$ and an IDB predicate Q, whether this program is functional for Q.

Proof: The proof is again by reduction from the containment of Datalog programs. Let Γ_1, Γ_2, Q, Q' and Γ_2' be like in the proof of Theorem 4.3. Let $stepone, steptwo$ be two new 0-ary predicates. Then consider the program $< \Gamma, \mathcal{S} >$ where $\mathcal{S}$ consists of the set of EDB in Γ_1 together with the predicate $stepone$, and Γ consists of the following N-Datalog* rules:

1. $head \leftarrow body, stepone$
   for each rule $head \leftarrow body$ in Γ_1.
2. $\neg stepone, steptwo \leftarrow$.
3. $head \leftarrow body, steptwo$
   for each rule $head \leftarrow body$ in Γ_2'.
4. $\neg Q(\vec{x}) \leftarrow Q'(\vec{x})$.

Note that the computation of Γ_1 can be interrupted nondeterministically at any stage by Rule 2. In particular, Rule 2 can always be applied before starting to apply Rule 1. Thus, for any input I, there exists a fixpoint of Γ with empty projection on Q.

Hence, the program Γ is functional for Q iff the last rule is always capable of erasing each tuple in Q. In particular, this must be true when Rules 1 have been saturated before applying Rule 2. Then, we have:

Claim: $\Gamma_1 \subseteq_Q \Gamma_2$ iff $< \Gamma, \mathcal{S} >$ is functional for Q. $\Box$

5 Determinism vs. Nondeterminism

In this section, we are concerned with programs that can be assigned both a deterministic and a nondeterministic semantics, i.e., in sets of rules with single literal heads and without occurrence of the equality predicate. Such sets of rules can be viewed as Datalog$^{\neg *}$ or as N-Datalog$^{\neg *}$ programs. Although nondeterministic programs can be functional, it is not necessarily true in that case that the nondeterministic semantics coincides with the deterministic semantics. The latter property is nonetheless interesting for an optimization purpose. Indeed, as discussed in [SdM88], implementing a nondeterministic program with deterministic semantics allows more efficient processing of the program. This is due to the fact that several instantiations of rules can be "fired" in parallel without changing the final result. In particular, for a given rule, the parallel firing of all its instantiations can be efficiently implemented using relational algebra operations. In this section, we study when the nondeterministic semantics of a program in the Datalog-like languages coincide with the deterministic semantics.

Obviously, for programs without negation, the deterministic and nondeterministic semantics coincide.

Let us consider now the Datalog$^{\neg}$ queries. The following example shows that functionality and coincidence of deterministic and nondeterministic semantics are distinct properties.

Example 5.1 Consider the query Γ consisting of the rules:

$A \leftarrow \neg A, \neg B,$

$B \leftarrow \neg A, \neg B,$

$C \leftarrow A, \neg B,$

$C \leftarrow \neg A, B$

$A \leftarrow C$

$B \leftarrow C$

This query, with nondeterministic semantics, is functional for all predicates. However, with the nondeterministic semantics, A, B, C are derived, whereas with the deterministic one, only A, B are. $\Box$

One can show that one cannot decide, given a query in Datalog$^\neg$ and N-Datalog$^\neg$, whether the deterministic and nondeterministic semantics coincide.

Theorem 5.2 It is undecidable, given a query Γ' in both N-Datalog$^\neg$ and Datalog$^\neg$ and a predicate T, whether for each instance I over the EDB predicates of Γ':

$$\{\ J[T] \mid (I,J) \in \Gamma_n'^\infty \} = \{\ \Gamma'^\infty(I)[T]\ \}.$$

Proof: Consider the construction of the query Γ in the proof of Theorem 4.3. Let T be a new 0-ary predicate. Then, consider the program Γ' consisting of Γ together with the rule:

$T \leftarrow R.$

One can easily check that T is always derived with the deterministic semantics and that it is derived with the nondeterministic semantics iff $\Gamma_1 \subseteq_Q \Gamma_2$. Thus the nondeterministic and the deterministic semantics coincide with respect to T iff the nondeterministic program Γ is functional for T iff $\Gamma_1 \subseteq_Q \Gamma_2$. $\Box$

Let us now consider the N-Datalog* case. In this case again, the deterministic and nondeterministic semantics may differ even for functional programs as shown by the following program:

Example 5.3 Consider the program $< \Gamma, \{A\} >$ consisting of the rules:

$\neg Q \leftarrow A,$

$Q \leftarrow A,$

$\neg A \leftarrow Q.$

This program, with nondeterministic semantics is functional for all predicates. Howewer, on input A, it yields $\{Q\}$ with the nondeterministic semantics and $\{A\}$ with the deterministic one. $\Box$

Indeed, for N-Datalog* *programs*, we have:

Theorem 5.4 It is undecidable, given a program $< \Gamma, \mathcal{S} >$ in both N-Datalog* and Datalog* and a predicate T, whether for each instance I over $\mathcal{S}$:

$$\{\ J[T] \mid (I,J) \in \Gamma_n'^\infty \} = \{\ \Gamma'^\infty(I)[T] \mid \Gamma'^\infty(I)[T] \text{ is defined } \}.$$

Proof: The proof resembles that of Theorem 5.2. Consider the construction of the program Γ in the proof of Theorem 4.4. Let T be a new 0-ary predicate. Consider the program Γ' consisting of Γ together with the rules:

$T \leftarrow Q(\vec{x})$,

$\neg T \leftarrow Q(\vec{x})$,

$\neg Q(\vec{x}) \leftarrow T$.

With the deterministic semantics, T is never derived. With the nondeterministic semantics, T is never derived iff Q is empty iff $\Gamma_1 \subseteq_Q \Gamma_2$. □

In the previous two proofs, we use the fact that N-Datalog$^{\neg}$ queries (respectively, N-Datalog* programs) are not always functional which is a major difference with the deterministic counterparts of these languages. Let us now consider the N-Datalog* queries. The same argument cannot be used here since such queries *are* functional.

As shown by the following example, the two semantics may differ also in this case.

Example 5.5 Consider the query Γ consisting of the rules:

$\neg Q \leftarrow$,

$Q \leftarrow$.

With the deterministic semantics, Q is never derived. With the nondeterministic one, the query loops forever. □

Although the two semantics may differ also in the case of N-Datalog* queries, we next show that one can detect when this happens. To prove it, we use a technical lemma that compares the two semantics.

Recall that an N-Datalog* query is functional. Thus each Γ_n^{∞} can be viewed as a function. We show that that function is closely related to Γ^{∞}.

Lemma 5.6 Let Γ be both a Datalog* and N-Datalog* query and R a predicate. Then for each I over the EDB predicates of Γ:

$$\text{if } \Gamma_n^{\infty}(I)[R] \text{ is defined, } \Gamma_n^{\infty}(I)[R] = \Gamma^{\infty}(I)[R].$$

Proof: the proof is a straightforward induction using Lemma 3.4. □

By the previous lemma, and using Theorem 3.2 that says that Datalog* queries are total, the two semantics coincide iff the N-Datalog* query is total which can be decided by Theorem 3.8. Thus, we have:

Theorem 5.7 It is decidable, given a query Γ in both N-Datalog* and Datalog*, whether the deterministic and nondeterministic semantics coincide. $\Box$

6 Avoiding loops

Although loops are inherently present as soon as "deletions" are introduced, there is a subtle way of avoiding them. A first illustration of this can be found in [AV87]. Loops are used there in a simulation of a procedural language by a declarative one. The case is made that loops can be "detected". We prove that this is the case, in a more fundamental way. More precisely, we introduce a notion of "loop-free simulation" of programs with loops and prove that for both deterministic and nondeterministic programs, such simulations always exist.

Let Γ be a program using predicates in $\mathcal{S}$. Let Γ' be a program using the predicates in $\mathcal{S}$ and a distinguished 0-ary predicate (not in $\mathcal{S}$), say *defined*. Then Γ' is a loop-free simulation of Γ if: on each input I over $\mathcal{S}$, Γ' always stops and:

- there is a non-terminating computation of Γ on input I iff there is a computation of Γ' on I which stops with *defined* false,
- Γ stops on input I with J as final state iff there is a computation of Γ' on input I which stops with *defined* true, and the restriction of the output to $\mathcal{S}$ is J.

Theorem 6.1 Each N-Datalog$^{\neg\star}$ program has a loop-free simulation in N-Datalog$^{\neg\star}$.

Proof: Intuitively, we implement a counter of computation steps. An overflow of the counter indicates the presence of a loop.

Let Γ be an N-Datalog$^{\neg\star}$ program. We obtain a loop-free simulation Γ' as follows. Let $P_1, ..., P_m$ be the predicates occurring in Γ. Let P be a new predicate with

$$arity(P) = N = \Sigma(arity(P_i)) + 1$$

and *order* be a predicate of arity 2. The predicate *order* will contain some arbitrary ordering of the constants in the input, say, $\{(a_0, a_1), (a_1, a_2), ..., (a_{n-2}, a_{n-1})\}$.

The loop-free simulation Γ' of Γ works as follows. First Γ' computes in *order* some arbitrary ordering of the constants occurring in the input instance. Based on this ordering a counter is implemented in relation P to count up to $2^{n^N} - 1$. This is done as follows. A tuple in P can be viewed as an N-digit number in base n, i.e. as an integer between 0 and $n^N - 1$. Now, the possible instances of P can be viewed as the subsets of $M = [0..n^N - 1]$. Let I be an instance over P with entries in $\{a_0,...,a_{n-1}\}$. Then I can be viewed as a set $\{i_1, ..., i_k\}$ of integers between 0 and $n^N - 1$. Furthermore, I can be viewed as representing the integer

$$\sum_{j=1}^{k} 2^{i_j} - 1.$$

This gives a bijection between instances over P with entries in $\{a_0, ..., a_{n-1}\}$ and $[0..2^{n^N} - 1]$. Indeed, one can effectively implement in N-Datalog$^{\neg\star}$ a counter between 0 and $2^{n^N} - 1$ in P. The lack of control due to nondeterminism is compensated by the existence of the ordering of the constants.

Now, Γ' alternates Γ-steps and counting-steps. The important point to notice is that the number of possible states reached in a computation of Γ is always less than $2^{n^N} - 1$ by construction of N. If the counter in Γ' reaches $2^{n^N} - 1$, then the computation in Γ entered some loop, and Γ' stops with *defined* false.

There is a subtlety in the use of *order*. Because of nondeterminism, one can never be sure that *order* contains all the constants occurring in I since control can be transferred prematurely to the simulation part. However, it can be ensured that Γ' will eventually detect that the counting is done on an incomplete ordering. In that case, the counter is reset to zero and *order* is expanded. In particular, such a checkpoint can be forced when Γ' believes that a loop has been encountered. Then, besides resetting the counter to zero, Γ' must set *defined* to true to acknowledge the fact that the detection of a loop may have been erroneous. □

Theorem 6.2 Each Datalog$^{\neg\star}$ program has a loop-free simulation in Datalog$^{\neg\star}$.

Proof: Intuitively, we carry on two identical computations in parallel. The computations are shifted by a fixed number of steps. When the two computations reach an identical state, we just have to check whether this state is a fixpoint. If this is not so, a loop has been detected.

Let Γ be a Datalog$^{\neg\star}$ program. Let Γ' be the program obtained by replacing each IDB predicate Q by a new predicate $\widehat{Q}$ (i.e., by "marking" each IDB predicate) in Γ. The loop-free simulation is realized by a program Γ'' as follows. Γ'' runs Γ and Γ' in parallel. Two steps of Γ' are simulated for each step of Γ. Intuitively, loops are detected by checking that Γ and Γ' reached the "same" state, i.e., that for each Q, Q and $\widehat{Q}$ have the same content. Suppose that on some input I, Γ enters in step M a loop of length K steps. Then for each $n \geq M$ and $j \geq 1$, $\Gamma^n(I) = \Gamma^{n+(j\dot{K})}(I)$. In particular, for $n = M\dot{K}$ and $j = M$,

$$\Gamma^{MK}(I) = \Gamma^{2MK}(I).$$

Now consider the computation of Γ''. When Γ has computed MK steps, Γ' has computed $2MK$ steps. At this point, Γ'' can detect that Γ and Γ' are in the same state,

More precisely, Γ'' performs the following:

while true do
    realize one iteration of Γ;
    realize two iterations of Γ';
    if Γ *and* Γ' *reached the same state*
    then   *if* Γ *reached a fixpoint*
        then make defined true and stop
        else make defined false and stop
endwhile

The control necessary for the above simulation can be implemented in Datalog$^{\neg\star}$.
□

The previous results suggest the following open problems:

Open problem: (i) Do all N-Datalog* programs have loop-free simulations in N-Datalog*? (ii) Do all Datalog* program have a loop-free simulation in Datalog*?

7 Conclusion

Some important properties of Datalog extensions of [AV88] have been studied. We showed that, unfortunately, the property of being functional, loop-free or total are lost in most cases and that these properties are in general undecidable. With respect to nondeterministic programs, the situation is even worse since even when the semantics is functional, one cannot guarantee that it coincides with the deterministic semantics.

Is the situation as bad as it looks?
We believe not. First, we exhibited sublanguages with at least some nice properties. (See the figure.) Also, we presented a technique for simulation of loops. (A similar technique for detection of "nonfunctionality" can be developed.) This suggests that although compile time detection of these properties is not feasible in general, run time detection is realistic.

The negative results that we presented and the importance (in our opinion) of the problems show that an important direction of research is to develop sufficient criteria for the properties. In that respect, constructions in the paper may provide useful guidelines for developing such criteria.

Acknowledgments: We thank Victor Vianu for very carefully reading a first draft of this paper and his many comments.

References

[AH88] S. Abiteboul and R. Hull. Data functions, Datalog and Negation. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 143–153, 1988.

[Apt87] K.R. Apt. Introduction to Logic Programming. Technical report, Department of Computer Sciences, Austin, Texas, 1987. To appear in Handbook of Theorical Computer Sciences, J. van Leeuwen.

[AV87] S. Abiteboul and V. Vianu. A Transaction Language Complete for Database Update and Specification. In *6th ACM Symposium on Principles of Database Systems*, pages 260–268, March 1987. To appear in Journal of Computer and System Sciences.

[AV88] S. Abiteboul and V. Vianu. Procedural and Declarative Database Update Language. In *7th ACM Symposium on Principles of Database Systems*, pages 240–250, March 1988.

[AV89] S. Abiteboul and V. Vianu. Fixpoint extensions of first-order logic and Datalog-like languages. In *Fourth IEEE Symposium on Logic in Computer Science*, Asilomar, California, 1989.

[BH86] N. Bidoit and R. Hull. Positivism vs. Minimalism in Deductive Databases. In *Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database System*, pages 123–132, 1986.

[BR86] F. Bancilhon and R. Ramakrishnan. An Amateur's Introduction to Recursive Query-Processing Strategies. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 16–52, 1986.

[CH80] A. Chandra and D. Harel. Computable Queries for Relational Data Bases. *Journal of Computer and System Sciences*, 21(2):pages 156–178, Oct. 1980.

[dMS88] C. de Maindreville and E. Simon. Modelling non-deterministic queries and updates in deductive databases. In *Proc. of Internat. Conf. on Very Large Databases*, Los Angeles, 1988.

[GS88] G. Gardarin and E. Simon. Les systemes de gestion de bases de donnees deductives. *Techniques et Sciences de l'Informatique*, 6:5, 1988.

[Kan88] P. Kanellakis. Elements of relational database theory. Technical report, Brown Univ., 1988. to appear as a chapter in Handbook of Theoretical Computer Science.

[KP87] P.G. Kolaitis and C.H. Papadimitriu. Why not negation by fixpoint ? In *7th ACM Symposium on Principles of Database Systems*, March 1987.

[SdM88] E. Simon and C. de Maindreville. Deciding whether a production rule is relational computable. In *Proc. 2nd Internat. Conf. on Database Theory*, Bruges, Belgium, 1988.

[Shm87] O. Shmueli. Decidability and expressiveness aspects of logic queries. In *Proc. 6th ACM Symp. on Principles of Database Systems*, pages 237–249, 1987.

[Ull88] J.D. Ullman. *Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.