Answering Recursive Queries under Keys and Foreign Keys is Undecidable

Diego Calvanese, Riccardo Rosati
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
{calvanese,rosati}@dis.uniroma1.it

Abstract

Query answering in the presence of integrity constraints is a fundamental problem in several settings, such as information integration. Keys, foreign keys and inclusion dependencies are the most common forms of constraints used in databases. It has been established recently that, in the presence of such constraints, query answering is decidable for non-recursive queries. Obviously, in the absence of constraints, query answering is also decidable for recursive queries, which are a powerful querying mechanism that subsumes query languages for semistructured data and the semantic web. It was open whether answering recursive queries in the presence of the above classes of constraints is decidable. In this paper we show that this is indeed not the case. In particular, we show that answering recursive queries under keys and foreign keys or under inclusion dependencies is undecidable, both for unrestricted and for finite databases.

1 Introduction

Integrity constraints are a fundamental modeling tool in databases [5, 2]. In many application domains, integrity constraints may be violated by the data collected and managed by the application. In principle, the issue of dealing with integrity constraint violations is relevant in all applications involving the integration of heterogeneous information (e.g., Semantic Web, Data Warehouses, Enterprise Resource Planning Systems). For instance, in data integration [8, 7], integrity constraints enrich the semantics of the global view of a set of autonomous information sources, while such constraints may be violated by data at the sources [6].

In the general case of a database in which data violate integrity constraints, the problem arises of how to interpret such a database. Traditionally, database theory adopts an *exact* interpretation of data, based on the *closed world assumption* [10], i.e., the interpretation of each relation r exactly corresponds to the extension of r in the database instance. In order to cope with data inconsistencies, other assumptions about data are adopted in the literature on information integration. In particular, many studies propose to adopt in such a setting a *sound* semantics [8], in which the interpretation of each relation r can be considered as a superset of the extension of r in the database instance.

In this paper we focus our attention on some of the most common forms of constraints used in databases, namely (*i*) single primary keys and foreign keys, and (*ii*) inclusion dependencies, and study query answering under the sound semantics.

It has been established recently that, in the presence of such constraints, query answering is decidable for conjunctive queries and for unions of conjunctive queries [3, 4]. Obviously, in the absence of integrity constraints (or in the presence of key dependencies only), query answering is also decidable for recursive queries [2], which are a powerful querying mechanism that subsumes query languages for semistructured data and the semantic web. It was open whether answering recursive queries in the presence of these classes of constraints is decidable.

We provide a negative answer to the above question. In particular, we develop our analysis both under the assumption that a database must be a *finite* structure, and under the assumption of *unrestricted* databases. We first prove that, in the presence of either inclusion dependencies or key and foreign key dependencies, answers to recursive queries over finite databases are in general different from the answers obtained over unrestricted databases, i.e., *finite controllability* does not hold for recursive query answering. Then, we prove that, both for unrestricted and for finite databases, recursive query answering is undecidable in the presence of inclusion dependencies or in the presence of key and foreign key dependencies. Finally, we extend our results to the problem of query containment under integrity constraints. In particular, we show that the problem of establishing whether a conjunctive query is contained into a recursive query, which is decidable in the absence of integrity constraints, is undecidable when either inclusion dependencies or key and foreign key dependencies are expressed over the database schema.

The above results imply that, under the sound semantics, the presence of even very simple forms of integrity constraints makes the problem of answering recursive queries (and the problem of deciding query containment of a conjunctive query w.r.t. a recursive query) undecidable. Conversely, in the same setting, conjunctive queries can be effectively answered by sound and complete algorithms (and containment between conjunctive queries can be effectively decided).

The paper is organized as follows. In Section 2 we recall the formal framework of relational databases with integrity constraints. Then, in Section 3 we study finite controllability of recursive query answering. In Section 4 we prove undecidability of recursive query answering both under inclusion dependencies and under key and foreign key dependencies. In Section 5 we extend the previous undecidability results to the problem of query containment under integrity constraints. Section 6 concludes the paper.

2 Framework

In this section we present the syntax and semantics of the relational model with integrity constraints. We assume that the reader is familiar with the basic notions of relational databases [2].

2.1 Syntax

We consider to have an infinite, fixed alphabet Γ of constants representing real world objects, and we take into account only database instances having Γ as domain. Moreover, we assume that different constants in Γ denote different objects, i.e., we adopt the so-called *unique name assumption*.

Basically, in the relational model we have to account for a set of relation symbols and a set of integrity constraints, i.e., assertions on the relation symbols that express conditions that are intended to be satisfied by database instances. In this paper we focus our attention on inclusion and key dependencies. More formally, we indicate a *relational schema* (or simply *schema*) $\mathcal{S}$ as a triple $\langle \mathcal{R}, \mathcal{I}, \mathcal{K} \rangle$, where:

- $\mathcal{R}$ is a relational alphabet, in which each relation has an associated arity that indicates the number of its attributes. The attributes of a relation r of arity n, are represented by the integers $1, \ldots, n$. We use r/n to denote that a relation r has arity n.

- $\mathcal{K}$ is a set of *key dependencies* (KDs), i.e., a set of assertions of the form $key(r) = \mathbf{A}$, where r is a relation in $\mathcal{R}$, and $\mathbf{A} = A_1, \ldots, A_n$ is a sequence of attributes of r. We assume that at most one key dependency is specified for each relation.

- $\mathcal{I}$ is a set of *inclusion dependencies* (IDs), i.e., a set of assertions of the form $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$, where r_1, r_2 are relations in $\mathcal{R}$, $\mathbf{A} = A_1, \ldots, A_n$ is a sequence of distinct attributes of r_1, and $\mathbf{B} = B_1, \ldots, B_n$ is a sequence of distinct attributes of r_2. Such an inclusion dependency is called a *foreign key* (FK) when there is a key dependency on r_2 with $key(r_2) = \mathbf{B}$.

In the following, we call *ID-schema* a schema of the form $\mathcal{S} = \langle \mathcal{R}, \mathcal{I}, \emptyset \rangle$, i.e., a schema containing only inclusion dependencies, while we call *FK-schema* a schema containing key and foreign key dependencies, i.e., a schema of the form $\mathcal{S} = \langle \mathcal{R}, \mathcal{I}, \mathcal{K} \rangle$ where all inclusions in $\mathcal{I}$ are foreign key dependencies.

A *relational query* (or simply *query*) over $\mathcal{S}$ is a formula that is intended to extract a set of tuples of constants of Γ. The language used to express queries over $\mathcal{S}$ is *Datalog* (without inequality), which we now define formally.

An *atom* is an expression of the form $r(x_1, \ldots, x_n)$ where r is a relation symbol of arity n and $x_1, \ldots, x_n$ are either variables or constants in Γ. A *rule* is an expression of the form

$$r(\vec{\mathbf{x}}) \leftarrow r_1(\vec{\mathbf{y}}_1), \ldots, r_k(\vec{\mathbf{y}}_k)$$

where $r(\vec{\mathbf{x}})$, $r_1(\vec{\mathbf{y}}_1), \ldots, r_k(\vec{\mathbf{y}}_k)$ are atoms, and each variable in $\vec{\mathbf{x}}$ appears among the variables in $\vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{y}}_k$. The atom $r(\vec{\mathbf{x}})$ is called the *head* of the rule, and $r_1(\vec{\mathbf{y}}_1), \ldots, r_k(\vec{\mathbf{y}}_k)$ is called its *body*. All variables are implicitly universally quantified outside the rule. A *Datalog query* q is a set of rules where at least one rule has q as relation symbol in the head. The arity n of the query is the arity of the relation symbol q. A *boolean query* is a query of arity 0. The relation symbols that occur in the heads of the rules are called *intensional* (IDB) symbols. The rest of the relation symbols are called *extensional* (EDB) symbols and must be among the symbols in $\mathcal{R}$.

Consider the graph G_q containing one node for each IDB symbol, and an edge from symbol r to symbol r' if r' appears in the body of a rule that has r as predicate in the head. The query q is said to be *recursive* if G_q contains a cycle.

2.2 Semantics

A *database instance* (or simply *database*) $\mathcal{B}$ for a schema $\mathcal{S} = \langle \mathcal{R}, \mathcal{I}, \mathcal{K} \rangle$ is a set of facts of the form $r(t)$ where r is a relation of arity n in $\mathcal{R}$ and t is an n-tuple of constants from Γ. We denote as $r^{\mathcal{B}}$ the set $\{t \mid r(t) \in \mathcal{B}\}$. A database $\mathcal{B}$ for a schema $\mathcal{S}$ is said to be *consistent* with $\mathcal{S}$ if it satisfies all the dependencies expressed on $\mathcal{S}$. In our framework, this means satisfying IDs in $\mathcal{I}$ and KDs in $\mathcal{K}$. More formally:

- $\mathcal{B}$ satisfies an inclusion dependency $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$ if for each tuple t_1 in $r_1^{\mathcal{B}}$ there exists a tuple t_2 in $r_2^{\mathcal{B}}$ such that $t_1[\mathbf{A}] = t_2[\mathbf{B}]$, where $t[\mathbf{A}]$ is the projection of the tuple t over $\mathbf{A}$. If $\mathcal{B}$ satisfies all inclusion dependencies expressed on $\mathcal{S}$, then we say that $\mathcal{B}$ is consistent with $\mathcal{I}$;

- $\mathcal{B}$ satisfies a key dependency $\mathit{key}(r) = \mathbf{A}$ if for each $t_1, t_2 \in r^{\mathcal{B}}$ with $t_1 \neq t_2$ we have $t_1[\mathbf{A}] \neq t_2[\mathbf{A}]$. If $\mathcal{B}$ satisfies all key dependencies expressed on $\mathcal{S}$ we say that $\mathcal{B}$ is consistent with $\mathcal{K}$.

Traditionally, in databases, the closed world assumption is made, i.e, given a schema $\mathcal{S}$ and a database $\mathcal{D}$, the set of tuples considered to be in a relation r are *exactly* those that $\mathcal{D}$ assign to r. Hence, $\mathcal{D}$ has to satisfy $\mathcal{S}$ and, when answering a query, one has to consider a single database only. On the other hand, one can adopt an open world semantics, and consider the tuples that $\mathcal{D}$ assigns to relations in $\mathcal{R}$ as a subset of the set of tuples that satisfy $\mathcal{S}$. In other words, the database $\mathcal{D}$ is considered to be a *sound* specification of the databases $\mathcal{B}$ for $\mathcal{S}$. Such an assumption is typical of settings with incomplete information, such as information integration, and it is the one we adopt here.

Formally, given a finite database $\mathcal{D}$ for $\mathcal{S} = \langle \mathcal{R}, \mathcal{I}, \mathcal{K} \rangle$, the semantics of $\mathcal{S}$ with respect to $\mathcal{D}$, denoted $\mathit{sem}(\mathcal{S}, \mathcal{D})$, is the set of (possibly infinite) databases $\mathcal{B}$ for $\mathcal{S}$ such that:

- $\mathcal{B}$ is consistent with $\mathcal{S}$, i.e., it satisfies the integrity constraints in $\mathcal{I}$ and $\mathcal{K}$;

- $\mathcal{B} \supseteq \mathcal{D}$.

It is easy to see that $\mathit{sem}(\mathcal{S}, \mathcal{D})$ contains several databases for $\mathcal{S}$. Moreover, we denote with $\mathit{sem}_f(\mathcal{S}, \mathcal{D})$ the subset of $\mathit{sem}(\mathcal{S}, \mathcal{D})$ constituted by all finite databases.

We give now the semantics of queries. For a rule

$$r(x_1, \ldots, x_n) \leftarrow r_1(\vec{\mathbf{y}}_1), \ldots, r_k(\vec{\mathbf{y}}_k)$$

the facts that can be deduced by applying the rule to a database $\mathcal{B}$ are the n-tuples $\langle c_1, \ldots, c_n \rangle$ of constants of Γ, such that there is a substitution of the variables in the body of the rule with constants of Γ that assigns c_i to x_i and such that, after the substitution, each atom of the body is in $\mathcal{B}$.

For a Datalog query q, given a database $\mathcal{B}'$ over both the extensional and the intensional relation symbols, let $\tau_q(\mathcal{B}')$ be the database obtained by adding to $\mathcal{B}'$ all the facts that are deduced by applying to $\mathcal{B}'$ all the rules in q. Then, the *evaluation* of q over a database $\mathcal{B}$, denoted as $ans(q, \mathcal{B})$, is the fixpoint of the application of $\tau_q(\cdot)$ starting from $\mathcal{B}$. Formally, let

$$\begin{array}{rcl} \mathcal{B}_0 & = & \mathcal{B} \\ \mathcal{B}_{i+1} & = & \tau_q(\mathcal{B}_i) \end{array}$$

Then $ans(q, \mathcal{B}) = q^{\mathcal{B}_h}$, where h is the least number such that $\mathcal{B}_h = \mathcal{B}_{h+1}$. Note that such an h always exists [2].

Finally, given a schema $\mathcal{S}$ and a database $\mathcal{D}$ for $\mathcal{S}$, we call *certain answers* to a Datalog query q of arity n with respect to $\mathcal{S}$ and $\mathcal{D}$, the set

$$cert(q, \mathcal{S}, \mathcal{D}) = \{\langle c_1, \ldots, c_n\rangle \mid \langle c_1, \ldots, c_n\rangle \in ans(q, \mathcal{B}),\ \text{for each } \mathcal{B} \in sem(\mathcal{S}, \mathcal{D})\ \}$$

Moreover, we consider also certain answers over finite databases, defined as follows:

$$cert_f(q, \mathcal{S}, \mathcal{D}) = \{\langle c_1, \ldots, c_n\rangle \mid \langle c_1, \ldots, c_n\rangle \in ans(q, \mathcal{B}),\ \text{for each } \mathcal{B} \in sem_f(\mathcal{S}, \mathcal{D})\ \}$$

Intuitively, the certain answers are the tuples that are answers to the query for every (finite) database in the semantics of the schema, and hence consistent with the available information [8].

3 Finite controllability

We start by studying finite controllability of the problem of answering recursive queries, i.e., we ask whether the answers to recursive queries over finite databases are in general different from the answers obtained over unrestricted databases.

The following theorem establishes that finite controllability does not hold for recursive query answering in the presence of inclusion dependencies.

Theorem 1 *There exist a Datalog query q, an ID-schema $\mathcal{S} = \langle \mathcal{R}, \mathcal{I}, \emptyset\rangle$, and a database $\mathcal{D}$ for $\mathcal{S}$ such that $cert(q, \mathcal{S}, \mathcal{D}) \neq cert_f(q, \mathcal{S}, \mathcal{D})$.*

Proof. Without loss of generality, we assume that the domain of constants Γ is of the form $\Gamma = \{n_1, n_2, \ldots, n_k, \ldots\}$. Now let $\mathcal{S}$ and $\mathcal{D}$ be as follows:

$$\begin{array}{rcl} \mathcal{R} & = & \{\, r/2 \,\} \\ \mathcal{I} & = & \{\, r[2] \subseteq r[1] \,\} \\ \mathcal{D} & = & \{\, r(n_1, n_2) \,\} \end{array}$$

and let q be the following boolean query, which makes use of an IDB predicate $t/2$:

$$\begin{array}{rcl} q & \leftarrow & t(X, X) \\ t(X, Y) & \leftarrow & r(X, Y) \\ t(X, Z) & \leftarrow & t(X, Y), r(Y, Z) \end{array}$$

One can easily verify that $\langle\rangle \in \mathit{cert}_f(q, \mathcal{S}, \mathcal{D})$, i.e., the boolean query q is true in all the databases belonging to $\mathit{sem}_f(\mathcal{S}, \mathcal{D})$. On the other hand, $\langle\rangle \notin \mathit{cert}(q, \mathcal{S}, \mathcal{D})$: in fact, for the infinite database $\mathcal{B}$ in $\mathit{sem}(\mathcal{S}, \mathcal{D})$ of the form

$$\mathcal{B} = \bigcup_{i=1}^{\infty} \{\, r(n_i, n_{i+1}) \,\}$$

we have that q is false in $\mathcal{B}$. $\square$

The above proof can be easily extended in order to show that finite controllability does not hold for recursive query answering also in the presence of key and foreign key dependencies. More precisely, we construct an FK-schema obtained from the previous schema $\mathcal{S}$ by adding the key dependency $\mathcal{K} = \{\, \mathit{key}(r) = 1 \,\}$, which makes the inclusion in $\mathcal{I}$ a foreign key dependency. Under this new definition of $\mathcal{S}$, it holds that $\langle\rangle \in \mathit{cert}_f(q, \mathcal{S}, \mathcal{D})$ while $\langle\rangle \notin \mathit{cert}(q, \mathcal{S}, \mathcal{D})$, which proves the following theorem.

Theorem 2 *There exist a Datalog query q, an FK-schema $\mathcal{S} = \langle \mathcal{R}, \mathcal{I}, \mathcal{K} \rangle$, and a database $\mathcal{D}$ for $\mathcal{S}$ such that $\mathit{cert}(q, \mathcal{S}, \mathcal{D}) \neq \mathit{cert}_f(q, \mathcal{S}, \mathcal{D})$.*

4 Query answering

We now show that the problem of determining whether a tuple is a certain answer (both over unrestricted and over finite databases) to a Datalog query with respect to a schema containing key and foreign key dependencies is undecidable. To do so, we exhibit a reduction from the halting problem for a deterministic Turing machine (TM).

Let $\mathcal{M} = (Q, \Sigma, q_0, \delta, q_h)$ be a deterministic TM, where:

- Q is the set of states of $\mathcal{M}$;
- Σ is the tape alphabet of $\mathcal{M}$, which contains a special blank symbol $\flat$;
- $q_o \in Q$ is the initial state of $\mathcal{M}$;
- q_h is the halting state of $\mathcal{M}$;
- $\delta : Q \times \Sigma \longrightarrow Q \times \Sigma \times \{\Leftarrow, \Rightarrow, \Leftrightarrow\}$ is the transition function of $\mathcal{M}$.

We assume that the machine works on a left-bounded, right-unbounded tape, and that initially all positions of the tape contain $\flat$.

Intuitively, the TM starts its computation in state q_0, with its head on the leftmost position of the tape. When the TM is in state q and its head is on a tape position containing the symbol s, then the machine makes a move according to $\delta(q, s) = (q_n, s_n, m)$, by switching to state q_n, replacing the symbol under the head with s_n, and moving its head according to what specified by m (the head moves one position to the left when m is $\Leftarrow$, one position to the right when m is $\Rightarrow$, and stays in place when m is $\Leftrightarrow$). We assume w.l.o.g. that $\mathcal{M}$ never moves past the leftmost position of the tape (e.g., it may initially write a special symbol # on the leftmost position, and then never move past #). We also assume that there are no transitions from the halting state q_h.

Formally, the state of the computation of $\mathcal{M}$, i.e., the tape contents, the head position, and the current state of $\mathcal{M}$, is described by a *configuration* of $\mathcal{M}$, which we represent as a finite string of $\Sigma^* \cdot (\Sigma \times Q) \cdot \Sigma^*$. The initial configuration is $\varepsilon(\flat, q_0)\varepsilon$. Given a configuration $w_\ell(s_1, q)s_2w_r$, with $w_\ell, w_r \in \Sigma^*$, if $\delta(s_1, q) = (s_n, q_n, \Rightarrow)$, then the next configuration is $w_\ell s_n(s_2, q_n)w_r$. Similarly for transitions that move the head to the left or leave it in place.

We say that the TM halts when, started on the leftmost position of an initially blank tape, it enters state q_h, i.e., the computation reaches a configuration of the form $w_\ell(s, q_h)w_r$ for some $s \in \Sigma$ and $w_\ell, w_r \in \Sigma^*$. The *halting problem* consists in deciding whether a given TM halts.

We encode TM computations following an idea in [9]. Specifically, given a TM $\mathcal{M} = (Q, \Sigma, q_0, \delta, q_h)$, we construct a schema $\mathcal{S}_{int} = \langle \mathcal{R}_{int}, \mathcal{I}_{int}, \mathcal{K}_{int} \rangle$ (which may be either an ID-schema or an FK-schema), a database $\mathcal{D}_{int}$, and a boolean Datalog query $q_\mathcal{M}$ such that $cert(q_\mathcal{M}, \mathcal{S}_{int}, \mathcal{D}_{int})$ is nonempty if and only if $\mathcal{M}$ halts on the initially blank tape. We first provide the reduction for query answering under keys and foreign keys over unrestricted databases. We discuss then how it can be adapted also for the other cases.

The set of relations is $\mathcal{R}_{int} = \{int/1, succ/2\}$. The key dependencies $\mathcal{K}_{int}$ are

$$\begin{array}{rcl} key(int) & = & \{1\} \\ key(succ) & = & \{1\} \end{array}$$

The foreign key dependencies $\mathcal{I}_{int}$ are

$$\begin{array}{rcl} int[1] & \subseteq & succ[1] \\ succ[2] & \subseteq & int[1] \end{array}$$

The database $\mathcal{D}_{int}$ contains just the fact

$$int(0)$$

The intention is that, in a database $\mathcal{B} \in sem(\mathcal{S}_{int}, \mathcal{D}_{int})$, the instances of *int* represent integers. We use such integers to identify both positions in a configuration and configurations. Then, we encode the computation of $\mathcal{M}$ by means of the query $q_\mathcal{M}$, making use of additional IDB predicate symbols *tape*/3 and *lt*/2. Intuitively, $tape(c, p, s)$ represents that, for the configuration identified by c, in the position identified by p, the symbol on the tape is s; instead $lt(i_1, i_2)$ means that i_1 represents an integer "smaller than" i_2, i.e., i_1 precedes i_2 according to the *succ* relation. Besides the constant 0, we make use of one constant s for each tape symbol s in Σ, and of one constant s^q for each pair $(s, q) \in \Sigma \times Q$.

We define $q_\mathcal{M}$ by means of the following rules.

- Rules defining the predicate *lt*:

$$\begin{array}{rcl} lt(I_1, I_2) & \leftarrow & succ(I_1, I_2) \\ lt(I_1, I_2) & \leftarrow & succ(I_1, I_3), lt(I_3, I_2) \end{array}$$

- Rules encoding the initial configuration of $\mathcal{M}$:

$$\begin{array}{rcl} \mathit{tape}(0,0,\flat_{q_0}) & \leftarrow & \\ \mathit{tape}(0,P,\flat) & \leftarrow & \mathit{lt}(0,P) \end{array}$$

- Rules for the transitions of $\mathcal{M}$, which encode how the symbols in the various positions of a configuration change when $\mathcal{M}$ moves according to a certain transition:

  - For each transition $\delta(q,s) = (q_n, s_n, \Leftrightarrow)$, the following rules:

$$\begin{array}{rcl} \mathit{tape}(C_1,P,s_n^{q_n}) & \leftarrow & \mathit{tape}(C,P,s^q), \mathit{succ}(C,C_1) \\ \mathit{tape}(C_1,P_1,S_1) & \leftarrow & \mathit{tape}(C,P,s^q), \mathit{succ}(C,C_1), \mathit{lt}(P_1,P), \mathit{tape}(C,P_1,S_1) \\ \mathit{tape}(C_1,P_1,S_1) & \leftarrow & \mathit{tape}(C,P,s^q), \mathit{succ}(C,C_1), \mathit{lt}(P,P_1), \mathit{tape}(C,P_1,S_1) \end{array}$$

    Intuitively, the first rule takes into account the change of the symbol under the head, while the second and third rules take into account that the remaining symbols, respectively to the left and the right of the head, do not change.

  - For each transition $\delta(q,s) = (q_n, s_n, \Leftarrow)$, the following rules:

$$\begin{array}{rcl} \mathit{tape}(C_1,P,s_n) & \leftarrow & \mathit{tape}(C,P,s^q), \mathit{succ}(C,C_1) \\ \mathit{tape}(C_1,P_2,S_2) & \leftarrow & \mathit{tape}(C,P,s^q), \mathit{succ}(C,C_1), \mathit{succ}(P_1,P), \mathit{lt}(P_2,P_1), \\ & & \mathit{tape}(C,P_2,S_2) \\ \mathit{tape}(C_1,P_1,S_1) & \leftarrow & \mathit{tape}(C,P,s^q), \mathit{succ}(C,C_1), \mathit{lt}(P,P_1), \mathit{tape}(C,P_1,S_1) \end{array}$$

    and for each symbol $s_1 \in \Sigma$, the following rule:

$$\begin{array}{rcl} \mathit{tape}(C_1,P_1,s_1^{q_n}) & \leftarrow & \mathit{tape}(C,P,s^q), \mathit{succ}(C,C_1), \mathit{succ}(P_1,P), \\ & & \mathit{tape}(C,P_1,s_1) \end{array}$$

    Intuitively, in this case, besides the change of the symbol under the head, the rules also take into account that the head moves to the left. More precisely, the first rule takes into account the change of the symbol under the head, the last set of rules the change of the symbol immediately to the left of the head (which becomes the symbol under the head in the new configuration), and the second and third rules encode that the remaining symbols, respectively to the left and to the right, do not change.

  - For each transition $\delta(q,s) = (q_n, s_n, \Rightarrow)$, a set of rules analogous to the ones of the previous case.

- Finally, to define the query predicate $q_{\mathcal{M}}$, for each symbol $s \in \Sigma$, the following rule:

$$q_{\mathcal{M}} \quad \leftarrow \quad \mathit{tape}(C,P,s_{q_h})$$

  These rules make the query true when $\mathcal{M}$ has reached the halting state q_h in some configuration.

Theorem 3 *The Turing machine $\mathcal{M}$ halts on the initially blank tape if and only if $\langle\rangle \in \mathit{cert}(q_{\mathcal{M}}, \mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$.*

Proof (sketch). "$\Rightarrow$" Let $\mathcal{C}$ be a halting computation of $\mathcal{M}$, and consider a database $\mathcal{B} \in \mathit{sem}(\mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$. Due to the key and foreign key dependencies, it will contain objects $o_0, o_1, \ldots \in \mathit{int}^{\mathcal{B}}$ such that $\langle o_i, o_{i+1}\rangle \in \mathit{succ}^{\mathcal{B}}$. Notice that, for $\mathcal{B}$, we have two possibilities:

1. The sequence $o_0, o_1, \ldots$ is infinite. In this case the domain of $\mathcal{B}$ is obviously infinite, and each configuration/position can be identified by a distinct integer. Hence, from the halting computation $\mathcal{C}$ of $\mathcal{M}$, we can obtain an instantiation of the rules in $q_{\mathcal{M}}$, which uses the objects $o_0, o_1, \ldots$ for the first and second components of the *tape* predicate, and that correctly encodes $\mathcal{C}$. From such an instantiation we get that $\langle\rangle \in \mathit{ans}(q_{\mathcal{M}}, \mathcal{B})$.

2. There are h and k, with $0 \leq h \leq k$, such that $\langle o_i, o_{i+1}\rangle \in \mathit{succ}^{\mathcal{B}}$, for $0 \leq i < k$, and $\langle o_k, o_h\rangle \in \mathit{succ}^{\mathcal{B}}$. In this case the domain of $\mathcal{B}$ is finite (at least the part relevant for identifying configurations/positions), and, if k is smaller than the number of configurations needed to reach the halting state, there will be a successive configuration/position that is associated to the same integer as the one associated to a preceding configuration/position. However, we still get an instantiation of the rules of $q_{\mathcal{M}}$ in such a way that $\langle\rangle \in \mathit{ans}(q_{\mathcal{M}}, \mathcal{B})$. In some sense, the "conflicts" due to two facts $\mathit{tape}(c, p, s_1)$ and $\mathit{tape}(c, p, s_2)$, with $s_1 \neq s_2$, "facilitate" finding such an instantiation.

"$\Leftarrow$" Since $\langle\rangle \in \mathit{cert}(q_{\mathcal{M}}, \mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$, we have in particular that $\langle\rangle \in \mathit{ans}(q_{\mathcal{M}}, \mathcal{B})$, where $\mathcal{B}$ is the database containing an infinite sequence $o_0, o_1, \ldots$ of objects in $\mathit{int}^{\mathcal{B}}$ such that $\langle o_i, o_{i+1}\rangle \in \mathit{succ}^{\mathcal{B}}$. When the rules of $q_{\mathcal{M}}$ are instantiated using such objects for the first and second components of *tape*, given that $\langle\rangle \in \mathit{ans}(q_{\mathcal{M}}, \mathcal{B})$, the tuples instantiating *tape* actually encode a halting computation of $\mathcal{M}$. $\Box$

The proof of Theorem 3 actually goes through almost unchanged if we build an ID-schema by removing from $\mathcal{S}_{\mathit{int}}$ the key dependencies in $\mathcal{K}_{\mathit{int}}$, and just consider the set $\mathcal{I}_{\mathit{int}}$ of inclusion dependencies (which, in this case, are not foreign keys):

$$\begin{array}{rcl} \mathit{int}[1] & \subseteq & \mathit{succ}[1] \\ \mathit{succ}[2] & \subseteq & \mathit{int}[1] \end{array}$$

The only difference is that, in this case, for an object $o \in \mathit{int}^{\mathcal{B}}$ there may be more than one object o' such that $(o, o') \in \mathit{succ}^{\mathcal{B}}$. However, such objects do not influence the line of reasoning in the proof above. Hence we get the following result.

Theorem 4 *Let $\mathcal{S}$ be either an FK-schema or an ID-schema. Then, the problem of computing the certain answers to a Datalog query over unrestricted databases with respect to $\mathcal{S}$ and a database $\mathcal{D}$ is undecidable.*

We now turn our attention to query answering over finite databases. Consider again the encoding of $\mathcal{M}$ by means of $q_{\mathcal{M}}$. From the proof of Theorem 3 it follows that, if $\mathcal{M}$ halts, then $\langle\rangle \in \mathit{ans}(q_{\mathcal{M}}, \mathcal{B})$ for all finite databases $\mathcal{B} \in \mathit{sem}_f(\mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$. On the other hand, even if $\mathcal{M}$ does not halt, it may still be the case that $\langle\rangle \in \mathit{ans}(q_{\mathcal{M}}, \mathcal{B})$ for all *finite* databases $\mathcal{B} \in \mathit{sem}_f(\mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$. This is due to the fact that, since $\mathcal{B}$ is finite, we are always in case 2 in the proof of Theorem 3, and hence it may be that a "conflict" in the encoding of configurations/positions always arises. To detect such situations we proceed as follows.

1. We construct a new query $q_{\mathcal{M}}^{\mathit{conf}}$, containing all rules of $q_{\mathcal{M}}$, except that the rules defining the query predicate $q_{\mathcal{M}}$ are replaced in $q_{\mathcal{M}}^{\mathit{conf}}$ by the following ones: for each pair of symbols s_1 and s_2 with $s_1 \neq s_2$, where s_i is either a symbol in Σ or a symbol s^q for some pair $(s, q) \in \Sigma \times Q$, the following rule:
$$q_{\mathcal{M}}^{\mathit{conf}} \quad \leftarrow \quad \mathit{tape}(C, P, s_1), \mathit{tape}(C, P, s_2)$$
These rules make $q_{\mathcal{M}}^{\mathit{conf}}$ true when there is a conflict in the encoding of configurations/positions.

2. We ask whether $\langle\rangle \in \mathit{cert}_f(q_{\mathcal{M}}^{\mathit{conf}}, \mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$. If the answer is *yes*, then we can conclude that $\mathcal{M}$ does not halt on the empty input string. Indeed, if $\mathcal{M}$ halted, there would be some finite model $\mathcal{B}_g \in \mathit{sem}_f(\mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$ containing no conflict, and hence we would have $\langle\rangle \notin \mathit{cert}(q_{\mathcal{M}}^{\mathit{conf}}, \mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$.

3. If the answer in the previous step was *no*, we can conclude that there is some finite model $\mathcal{B}_g \in \mathit{sem}_f(\mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$ containing no conflict. Then we ask whether $\langle\rangle \in \mathit{cert}_f(q_{\mathcal{M}}, \mathcal{S}_{\mathit{int}}, \mathcal{D}_{\mathit{int}})$. If the answer is *no* again, we can conclude that $\mathcal{M}$ does not halt. On the other hand, if the answer is *yes*, we can now conclude that $\mathcal{M}$ halts. Indeed, the evaluation of $q_{\mathcal{M}}$ on the finite model $\mathcal{B}_g$ produces no conflict, and hence correctly encodes a computation of $\mathcal{M}$ in which $\mathcal{M}$ reaches the accepting state.

It follows that, if we had a means of establishing whether a tuple is in the certain answer over finite databases, we could decide the halting problem. This leads us to our second main result.

Theorem 5 *Let $\mathcal{S}$ be either an FK-schema or an ID-schema. Then, the problem of computing the certain answers to a Datalog query over finite databases with respect to $\mathcal{S}$ and a database $\mathcal{D}$ is undecidable.*

Finally, notice that, for simplicity, in our encoding we used constants in the queries. We can easily remove constants from the query as follows:

1. introduce a new intensional predicate P_c for each constant c appearing in the rules of the query;

2. replace each constant c in a rule with a new variable X_c, and add the atom $P_c(X_c)$ to the body of the rule;

3. add for each constant c the fact $P_c(c)$ to the database $\mathcal{D}$.

Hence, the undecidability results in Theorems 4 and 5 hold also when the query contains no constants.

5 Query containment

In this section we analyze the problem of query containment under integrity constraints, and prove that the results for query answering obtained in the previous sections also hold for query containment.

We study the problem of query containment with respect to ID-schemas and FK-schemas, both over unrestricted databases and over finite databases. Formally, the problem is the following: given a schema $\mathcal{S}$, which may be either an ID-schema or an FK-schema, a conjunctive query q_1, and a Datalog query q_2, we want to decide whether:

1. q_1 is contained in q_2 w.r.t. $\mathcal{S}$ over unrestricted databases, denoted as $q_1 \sqsubseteq_{\mathcal{S}} q_2$, which corresponds to establish whether, for each database $\mathcal{D}$ for $\mathcal{S}$, $cert(q_1, \mathcal{S}, \mathcal{D}) \subseteq cert(q_2, \mathcal{S}, \mathcal{D})$;

2. q_1 is contained in q_2 w.r.t. $\mathcal{S}$ over finite databases, denoted as $q_1 \sqsubseteq^f_{\mathcal{S}} q_2$, which corresponds to establish whether, for each finite database $\mathcal{D}$ for $\mathcal{S}$, $cert_f(q_1, \mathcal{S}, \mathcal{D}) \subseteq cert_f(q_2, \mathcal{S}, \mathcal{D})$.

We now define a reduction of query answering to query containment. The reduction makes use of a well-known technique for reducing query answering to query containment (see, e.g., [1]), and extends it to the case when integrity constraints are expressed over the database schema.

Lemma 6 *Given a schema $\mathcal{S} = \langle \mathcal{R}, \mathcal{I}, \mathcal{K} \rangle$, a database $\mathcal{D} = \{f_1, \ldots, f_k\}$ containing k facts, a Datalog query q, and a tuple t, let $q_{\mathcal{D}}$ be the following conjunctive query:*

$$q_{\mathcal{D}}(t) \leftarrow f_1, \ldots, f_k$$

Then, $t \in cert(q, \mathcal{S}, \mathcal{D})$ if and only if $q_{\mathcal{D}} \sqsubseteq_{\mathcal{S}} q$, and $t \in cert_f(q, \mathcal{S}, \mathcal{D})$ if and only if $q_{\mathcal{D}} \sqsubseteq^f_{\mathcal{S}} q$.

By the above lemma, it is immediate to extend the results obtained in the previous sections to the query containment problem. In particular:

Theorem 7 *There exist a conjunctive query q_1, a Datalog query q_2, and an ID-schema (respectively, an FK-schema) $\mathcal{S}$ such that $q_1 \sqsubseteq^f_{\mathcal{S}} q_2$ and $q_1 \not\sqsubseteq_{\mathcal{S}} q_2$.*

Theorem 8 *Let $\mathcal{S}$ be either an FK-schema or an ID-schema, let q_1 be a conjunctive query and let q_2 be a Datalog query. Then, the problem of deciding whether $q_1 \sqsubseteq_{\mathcal{S}} q_2$ is undecidable.*

Theorem 9 *Let $\mathcal{S}$ be either an FK-schema or an ID-schema, let q_1 be a conjunctive query and let q_2 be a Datalog query. Then, the problem of deciding whether $q_1 \sqsubseteq^f_{\mathcal{S}} q_2$ is undecidable.*

The above results show that the problem of establishing whether a conjunctive query is contained into a Datalog query, which is decidable in the absence of integrity constraints, is undecidable when either inclusion dependencies or key and foreign key dependencies are expressed over the database schema.

6 Conclusions

In this paper we have shown that, under the sound semantics, recursive query answering is undecidable in the presence of inclusion dependencies or in the presence of key and foreign key dependencies, both over unrestricted and over finite databases. Moreover, we have shown that the problem of establishing whether a conjunctive query is contained into a recursive query is undecidable when either inclusion dependencies or key and foreign key dependencies are expressed over the database schema.

We point out that the reductions defined in Section 4 immediately imply that the undecidability results stated by Theorems 4 and 5 still hold if we restrict to ID-schemas with *unary* inclusion dependencies, i.e., inclusion dependencies that mention a single attribute, or to FK-schemas where key dependencies are unary. In other words, the presence of even extremely restricted forms of foreign keys and inclusion dependencies makes answering recursive queries an undecidable problem.

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS'98*, pages 254–265, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison Wesley Publ. Co., 1995.

[3] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. In *Proc. of CAiSE 2002*, volume 2348 of *LNCS*, pages 262–279. Springer, 2002.

[4] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS 2003*, pages 260–271, 2003.

[5] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *J. of Computer and System Sciences*, 28(1):29–59, 1984.

[6] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proc. of ICDT 2003*, pages 207–224, 2003.

[7] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[8] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.

[9] A. Y. Levy and M.-C. Rousset. Verification of knowledge bases based on containment checking. *Artificial Intelligence*, 101(1-2):227–250, 1998.

[10] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 119–140. Plenum Publ. Co., 1978.