Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Magic Sets for Data Integration*

Wolfgang Faber and **Gianluigi Greco** and **Nicola Leone**
Department of Mathematics, University of Calabria, Italy
{faber,ggreco,leone}@mat.unical.it

Abstract

We present a generalization of the Magic Sets technique to Datalog$^\neg$ programs with (possibly unstratified) negation under the stable model semantics, originally defined in (Faber, Greco, & Leone 2005; 2007). The technique optimizes Datalog$^\neg$ programs by means of a rewriting algorithm that preserves query equivalence, under the proviso that the original program is consistent. The approach is motivated by recently proposed methods for query answering in data integration and inconsistent databases, which use cautious reasoning over consistent Datalog$^\neg$ programs under the stable model semantics.

In order to prove the correctness of our Magic Sets transformation, we have introduced a novel notion of modularity for Datalog$^\neg$ under the stable model semantics, which is more suitable for query answering than previous module definitions, and which is also relevant per se. A module under this definition guarantees independent evaluation of queries if the full program is consistent. Otherwise, it guarantees soundness under cautious and completeness under brave reasoning.

Introduction

Datalog$^\neg$ programs are function-free logic programs where negation may occur in the bodies of rules. Datalog$^\neg$ with stable model semantics[1] (Gelfond & Lifschitz 1988) is a very expressive query language in a precise mathematical sense: Under brave (cautious) reasoning[2], Datalog$^\neg$ allows to express every query that is decidable in the complexity class NP (co-NP) (Schlipf 1995).

In many recent proposals for data integration and reasoning on inconsistent databases, query answering turned out to be co-NP-complete and, in fact, it was reduced to cautious reasoning on suitable Datalog$^\neg$ programs (Arenas, Bertossi, & Chomicki 2000; Greco, Greco, & Zumpano 2001; Lembo 2004; Bravo & Bertossi 2003). An important feature of these programs is that they are consistent, viz. that a stable model is guaranteed to exist. However, given the co-NP hardness of their evaluation, the design of optimization techniques is of utmost importance for applications in real scenarios, where the size of the input database may be huge.

In this paper, we focus on the optimization of Datalog$^\neg$ programs, by discussing an extension of the well-known Magic Set method (Bancilhon *et al.* 1986; Beeri & Ramakrishnan 1991). This method exploits the fact that while answering a user query, often only a certain part of the stable models needs to be considered, so there is no need to compute these models in their entirety. In fact, its aim is to focus the instantiation of the program to those ground rules that are really needed to answer the query, by propagating binding information from the query goal into the program rules. Differing from the original method, Datalog$^\neg$ requires also body-to-head propagation in the presence of unstratified negation. The key idea is then to identify rules for which this is necessary, which we term *dangerous rules*.

The formal properties of the proposed approach have been deeply analyzed. First, we show that the program obtained is query equivalent under brave and cautious reasoning to the original program if the latter is consistent, making it a perfect fit for data integration applications, where consistency is guaranteed. If the original program is not guaranteed to be consistent, we can still show that on the transformed program, brave reasoning is complete, and cautious reasoning is sound with respect to the original program.

In order to establish the above results, we introduce a suitable notion of modularity for query answering over Datalog$^\neg$ programs. Previous notions like splitting sets of (Lifschitz & Turner 1994) and modules of (Eiter, Gottlob, & Mannila 1997) have been defined for stable model generation, while the new notion is tailored to query answering.

Finally, we analyze the complexity of determining whether a predicate is dangerous in a given program, which is a central notion of our Magic Set method. It turns out that this task is NL-complete and thus tractable.

Datalog$^\neg$ Programs

An *atom* $p(t_1, \ldots, t_k)$ is composed of a predicate symbol p of *arity* k and terms $t_1, \ldots, t_k$, each of which is either a constant or a variable. A *literal* is either an atom a or its

*Supported by M.I.U.R. within projects "Potenziamento e Applicazioni della Programmazione Logica Disgiuntiva" and "Sistemi basati sulla logica per la rappresentazione di conoscenza: estensioni e tecniche di ottimizzazione," and "tocai.it: Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet."

[1]Unless explicitly specified, Datalog$^\neg$ will always denote Datalog with negation under stable model semantics in this paper.

[2]Note that brave and cautious consequences are also called possible and certain answers, respectively.

negation $\mathrm{not}\ a$. A (Datalog$^\neg$) *rule* r is of the form

$$h \texttt{ :- } b_1, \ldots, b_m, \mathrm{not}\ b_{m+1}, \ldots, \mathrm{not}\ b_n. \qquad (1)$$

where $h, b_1, \cdots, b_n$ are atoms and $0 \leq m \leq n$. The atom h is called the head of the rule. A Datalog$^\neg$ program is a set of Datalog$^\neg$ rules. If all defining rules of a predicate p are facts (that is, $n = m = 0$), then p is an *EDB predicate*; otherwise p is an *IDB predicate*. A set of facts for EDB predicates of a program $\mathcal{P}$ is called an *EDB* (for $\mathcal{P}$). A *query* $\mathcal{Q}$ is just an atom.

Let the (Herbrand) universe and base for a Datalog$^\neg$ program $\mathcal{P}$ be denoted $U_\mathcal{P}$ and $B_\mathcal{P}$, respectively. The ground instantiation of $\mathcal{P}$ w.r.t. $U_\mathcal{P}$ is denoted by $Ground(\mathcal{P})$. An interpretation is a subset of $B_\mathcal{P}$. A ground positive literal A (resp. negative literal $\mathrm{not}\ A$) is true w.r.t. I if $A \in I$ (resp. $A \notin I$); otherwise it is false. An interpretation I *satisfies* a ground rule $r \in Ground(\mathcal{P})$ if the head of r is true w.r.t. I whenever the body of r is true w.r.t. I. An interpretation I is a *model* of a Datalog$^\neg$ program $\mathcal{P}$ if I satisfies all rules in $Ground(\mathcal{P})$.

Each not -free program $\mathcal{P}$ has a least (under subset inclusion) model, which is denoted by $\mathrm{LM}(\mathcal{P})$ and is the unique *stable model* of $\mathcal{P}$. Given a Datalog$^\neg$ program $\mathcal{P}$ and an interpretation I, the *Gelfond-Lifschitz transform* $\mathcal{P}^I$ is obtained from $Ground(\mathcal{P})$ by deleting all rules containing $\mathrm{not}\ b$ where $b \in I$, and deleting all not literals in the remaining rules. The set of *stable models* of a Datalog$^\neg$ program $\mathcal{P}$, denoted by $\mathrm{SM}(\mathcal{P})$, is the set of interpretations I, such that $I = \mathrm{LM}(\mathcal{P}^\mathrm{I})$.

A program $\mathcal{P}$ is *consistent* if $\mathrm{SM}(\mathcal{P}) \neq \emptyset$, otherwise it is *inconsistent*. A program $\mathcal{P}$ is *data consistent* if $\mathcal{P}_\mathcal{F} = \mathcal{P} \cup \mathcal{F}$ is consistent for each EDB $\mathcal{F}$.

Given a ground atom a and a Datalog$^\neg$ program $\mathcal{P}$, a is a *cautious (or certain) consequence* of $\mathcal{P}$, denoted by $\mathcal{P} \models_c a$, if $\forall M \in \mathrm{SM}(\mathcal{P}) : \mathrm{a} \in \mathrm{M}$; a is a *brave (or possible) consequence* of $\mathcal{P}$, denoted by $\mathcal{P} \models_b a$, if $\exists M \in \mathrm{SM}(\mathcal{P}) : \mathrm{a} \in \mathrm{M}$. Given a query $\mathcal{Q}$, $Ans_c(\mathcal{Q}, \mathcal{P})$ denotes the set of substitutions ϑ, such that $\mathcal{P} \models_c \mathcal{Q}\vartheta$; $Ans_b(\mathcal{Q}, \mathcal{P})$ denotes the set of substitutions ϑ, such that $\mathcal{P} \models_b \mathcal{Q}\vartheta$.

Let $\mathcal{P}$ and $\mathcal{P}'$ be Datalog$^\neg$ programs and $\mathcal{Q}$ be a query. Then, $\mathcal{P}$ is *brave-sound* w.r.t. $\mathcal{P}'$ and $\mathcal{Q}$, denoted $\mathcal{P} \subseteq^b_\mathcal{Q} \mathcal{P}'$, if $Ans_b(\mathcal{Q}, \mathcal{P}_\mathcal{F}) \subseteq Ans_b(\mathcal{Q}, \mathcal{P}'_\mathcal{F})$ is guaranteed for each EDB $\mathcal{F}$; $\mathcal{P}$ is *cautious-sound* w.r.t. $\mathcal{P}'$ and $\mathcal{Q}$, denoted $\mathcal{P} \subseteq^c_\mathcal{Q} \mathcal{P}'$, if $Ans_c(\mathcal{Q}, \mathcal{P}_\mathcal{F}) \subseteq Ans_c(\mathcal{Q}, \mathcal{P}'_\mathcal{F})$ for all $\mathcal{F}$. $\mathcal{P}$ is *brave-complete* (resp., *cautious-complete*) w.r.t. $\mathcal{P}'$ and $\mathcal{Q}$, denoted $\mathcal{P} \supseteq^b_\mathcal{Q} \mathcal{P}'$ (resp., $\mathcal{P} \supseteq^c_\mathcal{Q} \mathcal{P}'$) if $Ans_b(\mathcal{Q}, \mathcal{P}_\mathcal{F}) \supseteq Ans_b(\mathcal{Q}, \mathcal{P}'_\mathcal{F})$ (resp., $Ans_c(\mathcal{Q}, \mathcal{P}_\mathcal{F}) \supseteq Ans_c(\mathcal{Q}, \mathcal{P}'_\mathcal{F})$). Finally, $\mathcal{P}$ and P' are *brave-equivalent* (resp., *cautious-equivalent*) w.r.t. $\mathcal{Q}$, denoted by $\mathcal{P} \equiv^b_\mathcal{Q} \mathcal{P}'$ (resp. $\mathcal{P} \equiv^c_\mathcal{Q} \mathcal{P}'$), if $\mathcal{P} \subseteq^b_\mathcal{Q} \mathcal{P}'$ and $\mathcal{P} \supseteq^b_\mathcal{Q} \mathcal{P}'$ (resp., $\mathcal{P} \subseteq^c_\mathcal{Q} \mathcal{P}'$ and $\mathcal{P} \supseteq^c_\mathcal{Q} \mathcal{P}'$).

Dangerous Rules and Independent Atom Sets

With every program $\mathcal{P}$, we associate a marked directed graph $DG_\mathcal{P} = (N, E)$, called the *predicate dependency graph* of $\mathcal{P}$, where (i) each predicate of $\mathcal{P}$ is a node in N, and (ii) there is an arc (a, b) in E directed from node a to node b if there is a rule $r \in \mathcal{P}$ such that two predicates a and b of literals appear in $B(r)$ and $H(r)$, respectively. Such an arc is marked if a appears in $B^-(r)$. A *cycle* of $DG_\mathcal{P}$ is a sequence of nodes $C = n_1, \ldots, n_k$, such that each n_i $(1 < i < k)$ occurs exactly once in C, $n_1 = n_k$, and each (n_i, n_{i+1}) $(1 \leq i < k)$ is an arc in $DG_\mathcal{P}$. An *odd cycle* in $DG_\mathcal{P}$ is a cycle $C = n_1, \ldots, n_k$ such that an odd number of the arcs (n_i, n_{i+1}) $(1 \leq i < k)$ is marked. In analogy, one can also define the *atom dependency graph* $DG^A_\mathcal{P}$ of a ground program $\mathcal{P}$, by considering atoms rather than predicates. We now use this notion to define dangerous predicates and rules. The intuition is that dangerous predicates may inhibit a stable model.

Definition 1 Let $\mathcal{P}$ be a program (resp., ground program), and d be a predicate (resp., atom) of $\mathcal{P}$. Then, we say that d is *dangerous* if either

1. d occurs in an odd cycle of $DG_\mathcal{P}$ (resp., $DG^A_\mathcal{P}$), or
2. d occurs in the body of a rule with a dangerous head predicate (resp., atom).

A rule r is *dangerous*, if it contains a dangerous predicate (resp., atom) in the head. □

Based on this definition, we define a notion of independence for sets of atoms. These sets must be closed under rules in the head-to-body direction and in the body-to-head direction for dangerous rules. The defining rules of these sets then form modules.

Definition 2 An *independent atom set* of a ground program $\mathcal{P}$ is a set $S \subseteq B_\mathcal{P}$ such that for each atom $a \in S$ the following holds:

1. if a is the head of a rule $r \in \mathcal{P}$ then all atoms of r are in S, and
2. if a appears in the body of a dangerous rule $r \in \mathcal{P}$ then all atoms of r are in S.

A subset T of a program $\mathcal{P}$ is a *module* if $T = \{r \mid \text{the head of } r \text{ is in } S\}$ for some independent atom set S. □

These modules can be used to partially evaluate programs, as the following results show.

Theorem 1 *Let T be a module of a ground program $\mathcal{P}$, then given an arbitrary EDB $\mathcal{F}$, the following holds:*

1. $\mathrm{SM}(\mathcal{P}_\mathcal{F})/_{\mathrm{T}_\mathcal{F}} \subseteq \mathrm{SM}(\mathrm{T}_\mathcal{F})$, *and*
2. $\mathrm{SM}(\mathrm{T}_\mathcal{F}) = \mathrm{SM}(\mathcal{P}_\mathcal{F})/_{\mathrm{T}_\mathcal{F}}$, *if $\mathcal{P}_\mathcal{F}$ is consistent.*[3]

Corollary 2 *Let T be a module of a ground program $\mathcal{P}$ and $\mathcal{F}$ be an EDB. If $\mathcal{P}_\mathcal{F}$ is consistent, each stable model of $\mathcal{P}_\mathcal{F}$ can be obtained by enlarging a stable model of $T_\mathcal{F}$.*

Importantly, this also means that one obtains the same answers to a query by considering only the module in which the query predicate is contained, under the proviso that the original program is consistent.

Theorem 3 *Given query $\mathcal{Q}$, which is covered by module T of a ground program $\mathcal{P}$, and an EDB $\mathcal{F}$, such that $\mathcal{P}_\mathcal{F}$ is consistent, then it holds that:*

1. $Ans_b(\mathcal{Q}, T_\mathcal{F}) = Ans_b(\mathcal{Q}, \mathcal{P}_\mathcal{F})$, *and*

[3] For an interpretation I and a set T of rules, $I/_T$ denotes the restriction of I to T, precisely, $I/_T = I \cap B_T$.

2. $Ans_c(\mathcal{Q}, T_{\mathcal{F}}) = Ans_c(\mathcal{Q}, \mathcal{P}_{\mathcal{F}})$.

More generally, the following properties hold for query answering:

Theorem 4 *Given query $\mathcal{Q}$, which is covered by module T of a ground program $\mathcal{P}$, it holds that:*

1. $T \supseteq^b_{\mathcal{Q}} \mathcal{P}$ *and* $T \subseteq^c_{\mathcal{Q}} \mathcal{P}$*:*
2. $T \equiv^b_{\mathcal{Q}} \mathcal{P}$ *and* $T \equiv^c_{\mathcal{Q}} \mathcal{P}$*, if $\mathcal{P}$ is data consistent.*

Magic Sets for Datalog$^\neg$ Programs

While with the notion of dangerous rules and modules we have an adequate means for identifying subprograms which are sufficient to answer a query, the definitions are not constructive. For this reason we next present a generalization of the Magic Set method for Datalog$^\neg$ programs, which effectively constructs a module for answering a given query.

The algorithm is given in Figure 1. Other than the traditional phased approach of Magic Sets, it uses a stack-based architecture, where the stack holds adorned predicates (predicates with an associated binding pattern, viz. one of b [bound] or f [free] for each argument). The idea is to create so-called magic predicates, which restrict the range of constants for predicate arguments to those relevant for the query. The relevant defining rules are then modified to include these magic predicates. Rules which are never reached from the query and are not dangerous will not be modified, and thus not considered.

The algorithm begins by analyzing the binding pattern of the query, pushing the respective adorned query predicate onto the stack and possibly storing some fact for the associated magic predicate (via the function ***BuildQuerySeeds***). After this initialization, the algorithm performs several steps for each adorned predicate by popping them from the stack (adorned predicates are pushed on the stack only once).

In steps *4-8*, the binding of $\mathtt{p}^\alpha$ is propagated head-to-body in each rule r of $\mathcal{P}$ having an atom $\mathtt{p}(\overline{\mathtt{t}})$ in the head. This propagation works as in the standard Magic Set method for stratified Datalog$^\neg$ programs using an appropriate sideways information passing strategy (SIPS).

Steps *9-13* perform the propagation of the binding through each *dangerous* rule d in $\mathcal{P}$ of the form $\mathtt{h}(\overline{\mathtt{t}_\mathtt{h}}) \texttt{ :- } \mathtt{Q}_1(\overline{\mathtt{t}_1}), \ldots, \mathtt{Q}_\mathtt{m}(\overline{\mathtt{t}_\mathtt{m}})$, in which the predicate $\mathtt{p}$ occurs in $\mathtt{Q}_\mathtt{i}(\overline{\mathtt{t}_\mathtt{i}})$ inside the positive or negative body. These steps are, in fact, crucial for guaranteeing cautious-completeness and brave-soundness for consistent programs. In this case, in order to simulate body-to-head propagations and to minimize the effort of doing so, we swap head and the matching body literal and apply the standard method as if it was a head-to-body propagation. So, the rule d is first replaced by an "inverted" rule d_s of the form $\mathtt{p}(\overline{\mathtt{t}_\mathtt{i}}) \texttt{ :- h}(\overline{\mathtt{t}_\mathtt{h}}), \mathtt{Q}_1(\overline{\mathtt{t}_1}), ..., \mathtt{Q}_{\mathtt{i}-1}(\overline{\mathtt{t}_{\mathtt{i}-1}}), \mathtt{Q}_{\mathtt{i}+1}(\overline{\mathtt{t}_{\mathtt{i}+1}}), ..., \mathtt{Q}_\mathtt{m}(\overline{\mathtt{t}_\mathtt{m}})$, which has been obtained by swapping the head atom with the body atom (possibly occurring negated) propagating the binding. Then, the adornment can be carried out as usual by means of the function ***Adorn***. Since this "inverted" rule was not part of the original program and its only purpose is generating binding information, it will not give rise to a modified rule, but only to magic rules. Finally, after all the adorned predicates have been processed the algorithm outputs the program $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$.

Input: A Datalog$^\neg$ program $\mathcal{P}$, and a query $\mathcal{Q} = \mathtt{g}(\overline{\mathtt{t}})$.
Output: The optimized program $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$.
var S: **stack** of adorned predicates; $modifiedRules$,$magicRules$: **set** of rules;
begin
1. $modifiedRules$:= $\emptyset$; $magicRules$:=***BuildQuerySeeds***($\mathcal{Q}$, S);
2. **while** $S \neq \emptyset$ **do**
3. p^α := S.**pop()**;
4. **for each** rule $r \in \mathcal{P}$ with $H(r) = p(\overline{t}_p)$ **do**
5. r_a := ***Adorn***(r,p^α,S);
6. $magicRules$:= $magicRules \bigcup$ ***Generate***(r_a);
7. $modifiedRules$:= $modifiedRules \bigcup \{$***Modify***$(r_a)\}$;
8. **end for**
9. **for each** dangerous rule $d \in \mathcal{P}$ of the form $\mathtt{h}(\overline{\mathtt{t}_\mathtt{h}}) :- \mathtt{Q}_1(\overline{\mathtt{t}_1}), \ldots, \mathtt{Q}_\mathtt{m}(\overline{\mathtt{t}_\mathtt{m}})$ where $Q_i = p$ or $Q_i = \text{not } p$ **do**
10. **let** d_s be the rule $\mathtt{p}(\overline{\mathtt{t}_\mathtt{i}}) :- \mathtt{h}(\overline{\mathtt{t}_\mathtt{h}}), \mathtt{Q}_1(\overline{\mathtt{t}_1}), \ldots, \mathtt{Q}_{\mathtt{i}-1}(\overline{\mathtt{t}_1}), \mathtt{Q}_{\mathtt{i}+1}(\overline{\mathtt{t}_1}), \ldots, \mathtt{Q}_\mathtt{m}(\overline{\mathtt{t}_\mathtt{m}})$;
11. **let** d_a:=***Adorn***(d_s,$\mathtt{p}^\alpha$,S);
12. $magicRules$:= $magicRules \bigcup$ ***Generate***(d_a);
13. **end for**
14. **end while**
15. $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$:=$magicRules \cup modifiedRules$;
16. **return** $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$;
end.

Figure 1: Magic Set Algorithm for Datalog$^\neg$ Programs.

We would like to show that $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$ forms a module of $\mathcal{P}$ which covers the query predicate. There is one technical issue, however, because the algorithm creates new symbols (the magic predicates). This means that $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$ formally is not a subset of $\mathcal{P}$ and hence cannot be a module by itself. But we can partially evaluate the program on the predicates that were introduced during the transformation, and consider the obtained equivalent ground program, which we can show to be indeed a module of $Ground(\mathcal{P})$. These considerations give rise to the following central result.

Theorem 5 *Let $\mathcal{P}$ be a Datalog$^\neg$ program, let $\mathcal{Q}$ be a query, and $\mathcal{F}$ be an EDB. Then, the following holds:*

1. $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P}) \subseteq^c_{\mathcal{Q}} \mathcal{P}$ *and* $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P}) \supseteq^b_{\mathcal{Q}} \mathcal{P}$*;*
2. $Ans_b(\mathcal{Q}, \mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})_{\mathcal{F}}) = Ans_b(\mathcal{Q}, \mathcal{P}_{\mathcal{F}})$*, if $\mathcal{P}_{\mathcal{F}}$ is consistent;*
3. $Ans_c(\mathcal{Q}, \mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})_{\mathcal{F}}) = Ans_c(\mathcal{Q}, \mathcal{P}_{\mathcal{F}})$*, if $\mathcal{P}_{\mathcal{F}}$ is consistent;*
4. $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P}) \equiv^b_{\mathcal{Q}} \mathcal{P}$ *and* $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P}) \equiv^c_{\mathcal{Q}} \mathcal{P}$*, if $\mathcal{P}$ is data consistent.*

Complexity Results

A desideratum of the Magic Set algorithm is of course that it runs in polynomial time in terms of the input program. All operations in the algorithm are deterministic and most are clearly simple. Moreover, under the realistic assumption that the maximum arity of predicates is bounded by a constant, it is also easy to see that the number of adorned predicates is bounded by a polynomial. There is however one step in the

algorithm that needs clarification, the cost of determining dangerous predicates and rules.

Theorem 6 *Let $\mathcal{P}$ be a program and d be a predicate. Then, deciding whether d is dangerous is* NL*-complete.*

This result is not straightforward, as deciding whether the first condition in Def. 1 holds is NP-complete. We conclude that the Magic Set algorithm is tractable for programs with bounded predicate arities. We can also specify the complexity for computing the set of dangerous rules.

Theorem 7 *Let $\mathcal{P}$ be a Datalog$^\neg$ program. All dangerous rules can be computed in $O(|Preds(\mathcal{P})|^3 + |\mathcal{P}|)$, where $Preds(\mathcal{P})$ is the set of IDB predicates of $\mathcal{P}$.*

An Application to Data Integration

This work had been motivated by needs arising within the project INFOMIX on data integration (Leone *et al.* 2005), funded by the European Commission. Here, a fully functional data integration system has been implemented by exploiting a reduction from query answering in data integration systems to cautious reasoning over Datalog$^\neg$ programs. Indeed, a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ (where $\mathcal{G}$ is a global schema, $\mathcal{S}$ are source schemata and $\mathcal{M}$ is a mapping) together with a database $\mathcal{D}$ for $\mathcal{S}$ and a query q over $\mathcal{G}$ are encoded into a Datalog$^\neg$ program $\Pi(q, \mathcal{I})$, in a way that cautious answers for $\Pi(q, \mathcal{I})$ correspond to the answers given by the data integration system, cf. (Lembo, Lenzerini, & Rosati 2003; Lembo 2004).

The binding propagation techniques proposed in this paper can be profitably exploited to isolate the relevant part of a database. Importantly, our optimization fits perfectly into the data integration framework. Indeed, the loosely-sound semantics for data integration always guarantees the existence of a database repair no matter of the types of constraints in Σ, provided that the schema is *non-key-conflicting* (Lembo 2004). Thus, the resulting logic program is consistent and our rewriting fully preserves the original semantics of the data-integration query, since query equivalence is ensured (Theorem 5).

Theorem 8 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, $\mathcal{D}$ be a database for $\mathcal{S}$, and q be a query over $\mathcal{G}$. Then, $ans(q, \mathcal{I}, \mathcal{D})$ coincides with $Ans_c(q, \mathtt{MS}^\neg(q, \Pi(q, \mathcal{I})) \cup \mathcal{D})$.*

In order to test the effectiveness of the Magic Set technique for query optimization in data integration systems, we have carried out some experiments on the demonstration scenario of the INFOMIX project, which refers to the information system of the University "La Sapienza" in Rome. A detailed discussion of the results is available in (INFOMIX Project Team 2004). The results confirmed that on various practical queries the performance is considerably improved by Magic Sets, even of order of magnitudes in some cases.

Our Magic Set technique provides benefits with respect to two crucial parameters in INFOMIX. By limiting the computation on the fraction of the retrieved global database which is relevant to the query binding, it generally produces smaller ground programs. Moreover, by disregarding mapping conflicts that are irrelevant for answering the query at hand, it can actually give rise to exponential savings.

Conclusion

We have provided a brief overview of an extension of the Magic Set method for Datalog$^\neg$ programs under the stable model semantics. The technique guarantees query equivalence for consistent programs, which occur in applications like data integration or querying inconsistent databases. Future work includes studying promising combinations with works such as (Bonatti 2004).

References

Arenas, M.; Bertossi, L. E.; and Chomicki, J. 2000. Specifying and querying database repairs using logic programs with exceptions. In *FQAS 2000*, 27–41.

Bancilhon, F.; Maier, D.; Sagiv, Y.; and Ullman, J. D. 1986. Magic Sets and Other Strange Ways to Implement Logic Programs. In *PODS'86*, 1–16.

Beeri, C., and Ramakrishnan, R. 1991. On the power of magic. *JLP* 10(1–4):255–259.

Bonatti, P. A. 2004. Reasoning with infinite stable models. *Artificial Intelligence* 156(1):75–111.

Bravo, L., and Bertossi, L. 2003. Logic programming for consistently querying data integration systems. In *IJCAI 2003*, 10–15.

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive Datalog. *ACM TODS* 22(3):364–418.

Faber, W.; Greco, G.; and Leone, N. 2005. Magic sets and their application to data integration. In *ICDT'05*, LNCS 3363

Faber, W.; Greco, G.; and Leone, N. 2007. Magic Sets and their Application to Data Integration. *JCSS* 73(4):584–609.

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *ICLP'88*, 1070–1080. Cambridge, Mass.: MIT Press.

Greco, G.; Greco, S.; and Zumpano, E. 2001. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *ICLP'01*, LNCS 2237

INFOMIX Project Team. 2004. Demo Scenario. Tech. Report INFOMIX S7-1, INFOMIX Project Consortium. `http://sv.mat.unical.it/infomix`.

Lembo, D.; Lenzerini, M.; and Rosati, R. 2003. Methods and techniques for query rewriting. Tech. Report D5.2, Infomix Consortium.

Lembo, D. 2004. *Dealing with Inconsistency and Incompleteness in Data Integration*. Ph.D. Dissertation, Universit di Roma "La Sapienza".

Leone, N.; Gottlob, G.; Rosati, R.; Eiter, T.; Faber, W.; Fink, M.; Greco, G.; Ianni, G.; Kałka, E.; Lembo, D.; Lenzerini, M.; Lio, V.; Nowicki, B.; Ruzzi, M.; Staniszkis, W.; and Terracina, G. 2005. The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In *SIGMOD 2005*, 915–917. ACM Press.

Lifschitz, V., and Turner, H. 1994. Splitting a Logic Program. In *ICLP'94*, 23–37. MIT Press.

Schlipf, J. S. 1995. The Expressive Powers of Logic Programming Semantics. *JCSS* 51(1):64–86.