MAGIC SETS WITH FULL SHARING

PAULO J AZEVEDO

▷ In this paper we study the relationship between tabulation and goal oriented bottom up evaluation of logic programs. Differences emerge when one tries to identify features of one evaluation method in the other. We show that to obtain the same effect as tabulation in top-down evaluation, one has to perform a careful *adornment* in programs to be evaluated bottom-up. Furthermore we propose an efficient algorithm to perform forward subsumption checking over adorned *magic facts*. ◁

1. Introduction

Much has been said about the relationship between goal oriented bottom-up and tabulated top-down evaluation of logic programs, see for instance [18, 17, 19, 11]. To give an example of these relations, we mention the equivalence between *magic facts* of bottom-up and *subgoals* in top-down evaluation. The order in which magic facts are derived is commonly referred to in the literature as the "order of subgoal evaluation".

Another example is the equivalence between the facts that can be derived by a specific magic fact and the stored solutions in memo tables for a specific subgoal. In the semi-naive procedure [2] a subsumption check is included which prevents the derivation of duplicate facts. Seki [16] observed that subsumption checking in this procedure has a counterpart in tabulated top-down evaluation in two ways:

- When subsumption is applied in magic facts it corresponds to the subsumption test of tabulation (admissibility test of SLD-AL).
- Subsumption applied in facts (non magic) derived during bottom-up evaluation corresponds to the duplicate elimination performed in tabulation when a new solution is inserted in the tables.

Address correspondence to Paulo J Azevedo, Departamento de Informatica, Universidade do Minho, Campus de Gualtar, 4710 Braga, Portugal, E-mail: pja@di.uminho.pt

THE JOURNAL OF LOGIC PROGRAMMING

©Elsevier Science Publishing Co., Inc., 1996
655 Avenue of the Americas, New York, NY 10010 0743-1066/93/$3.50

We investigate the use of subsumption, as described in the first case, to eliminate redundancy in the derivation of magic facts.

In this paper we are concerned with the recomputation that arises in the bottom-up evaluation of magic rewritten programs [4]. The starting point is to observe what features of tabulation appear in goal-oriented bottom-up evaluation. We will show that not all the desirable features of top-down appear entirely in bottom-up evaluation. Namely, we observe that a magic atom that does not subsume another magic atom can have its subgoal representative subsuming the subgoal representative of the latter. Since the adornment process yields syntactically different variants of the same predicate, the traditional implementation of subsumption cannot cope with adorned magic atoms. Consequently, the derivation of facts triggered by the magic fact $mag\_p^{bb}(a,a)$ is repeated by the derivation of facts triggered by $mag\_p^{bf}(a)$. Thus, goal-oriented bottom-up evaluation does not exhibit the full benefits of tabulation. We propose forms of overcoming this fault in Magic Sets by introducing two new techniques.

- First we suggest a different way of dealing with adornments.
- Secondly, we propose a new forward subsumption checking algorithm for detecting redundancy among adorned magic facts.

Although in this paper we only consider magic sets, future work will expand these proposals to more general techniques, e.g. *magic templates* [11].

2. Magic Sets & Tabulation

In the Magic Sets method [3] each rule has assigned a sideways information passing (SIP) strategy. This strategy represents a decision about the order in which the conditions of the rule will be evaluated and how values for variables are passed from conditions to other conditions during evaluation. There are two techniques for the implementation of these SIP strategies: One is the generation of magic rules. The other is the *adornment process*, where through a set of strings a representation of the expected pattern is attached to each literal. Adornments also ensure that range restriction is preserved.

Following [4], an *adornment* is a string from the alphabet {b,f} that represents the expected pattern of bound (b) and free (f) variables in the arguments of a predicate. Intuitively, an adorned occurrence of a predicate corresponds to a computation of that predicate with some arguments bound to constants and other arguments free. For instance, $\mathsf{p}^{\mathsf{bff}}$ corresponds to computing p with the first argument bound and the other two free. Notice that each SIP implicitly determines a pattern of bound/unbound arguments for each predicate to be evaluated. The task of the adornment process is to make this implicit pattern explicit.

As previously identified in the literature, e.g. [16, 18, 11], one consequence of the magic rewriting is that calls in top-down are represented in bottom-up by magic facts. In terms of adornments, the head of a magic rule has the same adornment as the literal that gives rise to the magic rule. Consequently, the adornments in a magic fact represent the pattern of bound and free arguments of a call.

Several authors observe that the well known features of tabulated top-down proof procedures also appear in goal-oriented bottom-up evaluaters. In [16], more detailed relations between these two forms of computation are put forward. The author identifies relations between SLD-AL [20], a tabulated proof procedure, and the Alexander Templates rewriting which is a variant of Supplementary Magic Sets without adornments. Magic predicates are *call* predicates in Alexander templates and derived facts are *sol* facts. Supplementary Magic Sets [4], avoid redundant joins by deriving supplementary relations. Seki establishes the relation between the admissibility test and the subsumption checking in a bottom-up evaluation. The latter subscribes the need in bottom-up evaluation (for instance in the semi-naive strategy) to check whether a newly derived fact is subsumed by a previously derived one. This subsumption checking can be reduced to simple duplicate elimination if only ground facts are derived. If a goal $\leftarrow \mathsf{q}$ is admissible (is not subsumed by a call stored in the tables) then correspondingly the subsumption checking in bottom-up determines that the fact $\mathsf{mag\_q}$ (or $\mathsf{call\_q}$ in Alexander templates) is a newly derived one. On the other hand a newly derived lemma L in SLD-AL corresponds to a newly derived fact in Alexander Templates sol_L or simply L in Magic Sets. The conclusion that one should draw from these two remarks is that subsumption checking in bottom-up has a counterpart in top-down in two ways. First, in the admissibility test on calls and second in the duplicate elimination performed on lemmas. However, one should notice that the introduction of adornments can corrupt these relations. Consider the case where the fact $\mathsf{mag\_q}$ was previously derived and it is 'compared' with the fact $\mathsf{mag\_q^b}$. Since syntactically they are unrelated, one cannot establish any subsumption relation between the two magic facts.

It is interesting to notice that tabulation systems like XSB [15] do not incorporate a subsumption checking mechanism but rather perform variant checking[1]. For reasons related to the way the answers to a call (and the stored call) are indexed, XSB uses a much simpler method to eliminate redundancy. Apart from efficiency purposes there are other reasons for checking for identity based on variance. Among them one can include the combination of negation and tabulation and the support of meta-programming facilities [5]. The price to pay is that not all recomputation is eliminated. For instance if the call $?\mathsf{p}(\mathsf{a},\mathsf{Y})$ is stored then only variants of this call e.g. $?\mathsf{p}(\mathsf{a},\mathsf{Z})$ are considered as having their answers in the tables. Thus if a call that is an instance of our stored one is derived e.g. $?\mathsf{p}(\mathsf{a},\mathsf{b})$ it is not identified as having answers in the table and consequently is recomputed in the program. Notice however that this phenomenon is equivalent to the one that arises within magic sets and semi-naive evaluation. The calls $?\mathsf{p}(\mathsf{a},\mathsf{Y})$ and $?\mathsf{p}(\mathsf{a},\mathsf{Z})$ are equivalent to the magic fact $\mathsf{mag\_p^{bf}}(\mathsf{a})$. Thus, when the latter call (magic fact) is derived it is identified as already answered. Consequently it is eliminated by the subsumption checking performed on derived facts by the semi-naive procedure. However, the second described case corresponds in magic sets to derive first the magic fact $\mathsf{mag\_p^{bf}}(\mathsf{a})$ and then $\mathsf{mag\_p^{bb}}(\mathsf{a},\mathsf{b})$. Applying subsumption checking between these two facts returns failure because they are syntactically unrelated. Therefore, the magic fact $\mathsf{mag\_p^{bb}}(\mathsf{a},\mathsf{b})$ is derived and the computation associated with it is redone. The aim of the following section is to explore the details involved with adornments and the desirable feature of sharing answers among computations of related magic facts.

[1]One atom is a *variant* of another if they are the same up to variable renaming.

3. Adornments and 'Sharing'

Consider in top-down evaluation the following order of calls: first $?\mathsf{p}(\mathsf{X},\mathsf{Y})$ and then $?\mathsf{p}(\mathsf{a},\mathsf{Z})$. In tabulated top-down the admissibility (subsumption) test would force the second call to reuse the answers that were computed and stored by the first. In the bottom-up evaluation of the corresponding magic rewritten program, the fact $\mathsf{mag\_p}^{\mathsf{ff}}$ would be generated first which would lead to the computation of the complete extension of predicate p i.e. all its solutions. Then, $\mathsf{mag\_p}^{\mathsf{bf}}(\mathsf{a})$ would be generated leading to the computation of p facts that have a as first parameter. Notice that, according to the described subgoals, the magic fact $\mathsf{mag\_p}^{\mathsf{bf}}(\mathsf{a})$ is redundant in relation to $\mathsf{mag\_p}^{\mathsf{ff}}$: the facts 'computed' by $\mathsf{mag\_p}^{\mathsf{bf}}(\mathsf{a})$ are included in the facts 'computed' by $\mathsf{mag\_p}^{\mathsf{ff}}$.

In magic adorned programs, apart from duplicate elimination, subsumption must also prevent redundant computation by eliminating the derivation of redundant magic facts. However, subsumption does not work on adorned programs because syntactically differently adorned magic facts are unrelated. Furthermore, following the earlier example, the computed answers for the predicate p^{ff} cannot be shared with predicate p^{bf} since both are now different predicates. Thus due to adornments, in semi-naive evaluation of magic programs, the subsumption test cannot check that answers derived with the first magic fact should be used to answer the requirements of the second fact. In this way, adornments remove from bottom-up one of the most desirable features of tabulation: the sharing of answers between similar calls.

Our aim is to have a bottom-up evaluation that preserves the sharing of solutions among common calls, as it happens in top-down evaluation. To achieve this, a program will be adorned in a different way. Adornments are used in several query optimization techniques, helping to cut down the relevant search space, e.g. [10, 6, 8]. But for the magic rewriting it is only necessary to consider adornments in the magic literals. In this way all adorned versions of a predicate will generate answers that potentially can be used by all the different adorned literals present in the body of rules. An implicit adornment is considered instead of an explicit 'renaming' of literals in rules. Consider the following rule.

$$\mathsf{p}(\mathsf{X},\mathsf{Y}) \leftarrow \mathsf{a}(\mathsf{X},\mathsf{Z}) \,\&\, \mathsf{b}(\mathsf{Y},\mathsf{Z})$$

Assuming the query $?\mathsf{p}(\mathsf{a},\mathsf{Y})$ and a left-to-right SIP strategy, standard adornment together with magic rewriting produces:

$$\mathsf{p}^{\mathsf{bf}}(\mathsf{X},\mathsf{Y}) \leftarrow \mathsf{a}^{\mathsf{bf}}(\mathsf{X},\mathsf{Z}) \,\&\, \mathsf{b}^{\mathsf{fb}}(\mathsf{Y},\mathsf{Z}) \,\&\, \mathsf{mag\_p}^{\mathsf{bf}}(\mathsf{X})$$

However, according to our idea of implicit adornments, it is sufficient to adorn the magic predicate only, which leads to:

$$\mathsf{p}(\mathsf{X},\mathsf{Y}) \leftarrow \mathsf{a}(\mathsf{X},\mathsf{Z}) \,\&\, \mathsf{b}(\mathsf{Y},\mathsf{Z}) \,\&\, \mathsf{mag\_p}^{\mathsf{bf}}(\mathsf{X})$$

On generating magic rules the same idea applies. For instance, the magic rules for a and for b are:

$$\mathsf{mag\_a}^{\mathsf{bf}}(\mathsf{X}) \leftarrow \mathsf{mag\_p}^{\mathsf{bf}}(\mathsf{X})$$

$$\mathsf{mag\_b}^{\mathsf{fb}}(\mathsf{Z}) \leftarrow \mathsf{mag\_p}^{\mathsf{bf}}(\mathsf{X}) \,\&\, \mathsf{a}(\mathsf{X},\mathsf{Z})$$

In this way we gain a generation of facts of the same predicate that enables the sharing of answers between literals of the same predicate in the body of rules.

A single rule can still generate several different adorned versions as happens in the standard adornment, since the information provided by the SIP strategy is still followed. Furthermore the main aim of adornments is still considered i.e. to implement the SIP strategy. Observe that by omitting adornments from literals in the body of rules we do not lose the benefits provided by the adornment process. The adornment is implicit in the way that rules are processed.

Combined with this new rewriting we need a subsumption test on the generated magic facts capable of identifying redundant magic facts. In the next section, an efficient algorithm for performing this task will be described.

4. Subsumption Checking over Adorned Atoms

Consider the following example, which is a recursive definition of $\mathsf{ancestor}$.

$\mathsf{anc}(X, Y) \leftarrow \mathsf{par}(X, Y).$
$\mathsf{anc}(X, Y) \leftarrow \mathsf{par}(X, Z)\ \&\ \mathsf{anc}(Z, Y).$

The predicate par corresponds to the following chain: $a \longrightarrow b \longrightarrow c \longrightarrow d$, which is the EDB:

$\mathsf{par}(a, b).$
$\mathsf{par}(b, c).$
$\mathsf{par}(c, d).$

The transformed program according to the standard adornment process and the query anc^{fb} is:

$\mathsf{anc}^{fb}(X, Y) \leftarrow \mathsf{par}(X, Y)\ \&\ \mathsf{mag\_anc}^{fb}(Y).$
$\mathsf{anc}^{fb}(X, Y) \leftarrow \mathsf{par}(X, Z)\ \&\ \mathsf{anc}^{bb}(Z, Y)\ \&\ \mathsf{mag\_anc}^{fb}(Y).$
$\mathsf{mag\_anc}^{bb}(Z, Y) \leftarrow \mathsf{par}(X, Z)\ \&\ \mathsf{mag\_anc}^{fb}(Y).$

$\mathsf{anc}^{bb}(X, Y) \leftarrow \mathsf{par}(X, Y)\ \&\ \mathsf{mag\_anc}^{bb}(X, Y).$
$\mathsf{anc}^{bb}(X, Y) \leftarrow \mathsf{par}(X, Z)\ \&\ \mathsf{anc}^{bb}(Z, Y)\ \&\ \mathsf{mag\_anc}^{bb}(X, Y).$
$\mathsf{mag\_anc}^{bb}(Z, Y) \leftarrow \mathsf{par}(X, Z)\ \&\ \mathsf{mag\_anc}^{bb}(X, Y).$

If the query is $?\mathsf{anc}(X, d)$ then we add the magic fact $\mathsf{mag\_anc}^{fb}(d)$. The semi-naive evaluation is:

$$T^1 = EDB \cup \{\mathsf{mag\_anc}^{fb}(d)\}$$

$$T^2 = T^1 \cup \{\mathsf{anc}^{fb}(c, d), \mathsf{mag\_anc}^{bb}(b, d), \mathsf{mag\_anc}^{bb}(c, d), \mathsf{mag\_anc}^{bb}(d, d)\}$$

$$T^3 = T^2 \cup \{\mathsf{anc}^{bb}(c, d)\}$$

$$T^4 = T^3 \cup \{\mathsf{anc}^{bb}(b, d), \mathsf{anc}^{fb}(b, d)\}$$

$$T^5 = T^4 \cup \{\mathsf{anc}^{fb}(a, d)\}$$

Notice that all the derived magic facts in step 2 (T^2) are redundant in relation to the magic fact representing the initial query, because they are subsumed by the latter. Furthermore, the $\mathsf{anc}^{\mathsf{fb}}$ fact derived at step 2 cannot be used by the $\mathsf{anc}^{\mathsf{bb}}$ literal in the body of the second rule defining $\mathsf{anc}^{\mathsf{fb}}$. Although syntactically $\mathsf{anc}^{\mathsf{bb}}(\mathsf{c},\mathsf{d})$ is different from $\mathsf{anc}^{\mathsf{fb}}(\mathsf{c},\mathsf{d})$, both represent that c is an ancestor of d.

A new algorithm is required to identify subsumption relations between the adorned magic facts. This is analogous to the *admissibility test* for top-down tabulation referred to in [20]. As shown in the example, although syntactically unrelated, semantically (based on the information contained in the adornments) one adorned magic fact can subsume another. For instance, the magic fact $\mathsf{mag\_p}^{\mathsf{bff}}(\mathsf{a})$ subsumes the fact $\mathsf{mag\_p}^{\mathsf{bbf}}(\mathsf{a},\mathsf{b})$, since the former corresponds to a goal $?\mathsf{p}(\mathsf{a},\mathsf{Y},\mathsf{Z})$ and the latter to $?\mathsf{p}(\mathsf{a},\mathsf{b},\mathsf{X})$. Without such a subsumption test, the full benefits associated with tabulation cannot be obtained in bottom-up evaluation.

4.1. A new definition of subsumption

First, we define subsumption in adorned magic facts. We rely on a translation from magic facts into the corresponding subgoals in top down evaluation. Since we are dealing with magic sets, we assume that no *aliasing* of variables [19] occurs (i.e. all magic facts represent atoms with distinct variables) and no derived fact contains function symbols in its arguments.

Definition 4.1. The translation of a magic fact $mag\_S^{\alpha}(\vec{c})$, where α is the adornment sequence of 'b's and 'f's, is the term $S(\vec{x})$ where $\vec{x}$ is composed of the constants that appear in $\vec{c}$ for the parameters that are 'b' in α and a distinct variable for each parameter that are 'f' in α.

For instance, the magic fact

$$mag\_p^{bfbbf}(a,b,c)$$

is translated into the term

$$?p(a,X,b,c,Y).$$

Subsumption between two magic facts is reduced to the subsumption between the corresponding subgoals resulting from the translation described above.

Definition 4.2. A magic fact M_1 subsumes a magic fact M_2 if the corresponding term S_1 of M_1 subsumes the term S_2 of M_2 i.e. $S_1 \sqsupseteq S_2$.

The idea is that instead of translating adorned magic facts into corresponding atoms and checking subsumption between these atoms, one can make use of the information in the adornments to directly determine whether an adorned magic fact subsumes another. Since adornments in magic facts represent the pattern of bound/free variables in their arguments, subsumption checking can be reduced to operations over adornment sequences ('bf' sequences).

Recall that G subsumes S (denoted $G \sqsupseteq S$) if there is a substitution θ for the variables in G such that $G\theta = S$. Thus, the subsumption test should check if such a substitution θ exists, succeeding if it does, failing otherwise. An alternative way to define subsumption is the following [9]:

Definition 4.3. $G \sqsupseteq S$ if $\exists \theta = m.g.u(G, S)$ and $S\theta = S$.

This is equivalent to say that G subsumes S if the most general unifier (m.g.u) of S and G does not bind any variable in S. Considering definition 4.3 of subsumption we can think of subsumption checking as reduced to operations with arguments of the atoms to be checked. Notice that since we are dealing with magic sets, all the described programs are Datalog and no *aliasing* of variables occurs. Removing aliasing is straightforward through the program transformation proposed in [19]. To optimize the operation with adornments we translate the 'bf' adornment sequences.

Definition 4.4. The translation of an adornment sequences is a binary number obtained through the following substitution:

- each position '*b*' in the original adornment is substituted by the digit '1'.
- and each '*f*' by the digit '0'.

Thus, an initial adornment has now a translation into a sequence of bits (binary number), e.g. the sequence bfbf is translated into the sequence of bits '1010'. The advantage of such a translation is that one can reduce the operations over arguments that occur in subsumption checking into logical operations on bits i.e. logical operations with binary numbers. For convenience and since we are operating with the adornment sequences, each adornment is an extra argument of the corresponding magic fact. For instance, the original magic fact $mag\_p^{bfbf}(c, a)$ is now the term $mag\_p(1010, c, a)$ where the adornment sequence is the first argument of the magic fact. The full new rewriting can now be presented:

Definition 4.5. Let P^{ad} be the adorned version of program (database) P following a given SIP-strategy and a query $\mathsf{q}(\vec{\mathsf{x}})$.

1. create a new predicate $\mathsf{mag\_p}(\mathsf{bit}, \vec{\mathsf{t}}^{\mathsf{b}})$ for each $\mathsf{p}^{\mathsf{ad}}(\vec{\mathsf{t}})$ in P^{ad}; $\vec{\mathsf{t}}^{\mathsf{b}}$ means the bound arguments of $\vec{\mathsf{t}}$ and bit is the translation of the adornment ad following definition 4.4.
2. for each rule in P^{ad} add the modified rule to $\mathsf{P}^{\mathsf{magic}}$ which is the original rule with the body extended with the literal $\mathsf{mag\_p}(\mathsf{bit}, \vec{\mathsf{t}}^{\mathsf{b}})$ if the head is $\mathsf{p}^{\mathsf{ad}}(\vec{\mathsf{t}})$ (i.e. only the bound (b) arguments are in the magic literal).
3. For each rule $\mathsf{p}^{\mathsf{ad}_0}(\vec{\mathsf{t}}) \leftarrow \mathsf{q}_1^{\mathsf{ad}_1}(\vec{\mathsf{t}}_1)$ & ... & $\mathsf{q}_{\mathsf{n}}^{\mathsf{ad}_{\mathsf{n}}}(\vec{\mathsf{t}}_{\mathsf{n}})$ in P^{ad} generate several magic rules:
   $\mathsf{mag\_q}_{\mathsf{i}}(\mathsf{bit}_{\mathsf{i}}, \vec{\mathsf{t}}_{\mathsf{i}}^{\mathsf{b}}) \leftarrow \mathsf{mag\_p}(\mathsf{bit}_0, \vec{\mathsf{t}}^{\mathsf{b}})$ & $\mathsf{q}_1(\vec{\mathsf{t}}_1)$ & ... & $\mathsf{q}_{\mathsf{i}-1}(\vec{\mathsf{t}}_{\mathsf{i}-1})$ is added to $\mathsf{P}^{\mathsf{magic}}$ for each $1 \leq \mathsf{i} \leq \mathsf{n}$ and the order of i respects the order on the SIP. Again $\mathsf{bit}_{\mathsf{i}}$ is the translation of the adornment ad_{i},
4. add the seed fact $\mathsf{mag\_q}(\mathsf{bits}, \vec{\mathsf{x}}^{\mathsf{b}})$ representing the query $\mathsf{q}(\vec{\mathsf{x}})$, where bits is the translation of the adornment associated with $\vec{\mathsf{x}}$.

The following theorems are stated without proof. Full proofs can be found in [1].

Theorem 4.1. (Preservation of answers) Let $< p^a, P^{ad} >$ *be a query and an adorned program transformed by standard magic sets rewriting [4]. Let* $< p^b, P^{bits} >$ *be the same query and program transformed following definition 5.* $< p^a, P^{ad} >$ *and* $< p^b, P^{bits} >$ *are equivalent i.e. the two programs produce the same answer for the resulting queries on p.*

This can be proved by considering the correspondence between the binary numbers and the adornment sequences.

Theorem 4.2. (Efficiency) Let P be a program and q a query. Let P^{mg} *be P and q with the original magic rewriting applied [4]. Let* P^{bits} *be P and q with the rewriting of definition 5 applied. Let* $\mathcal{S}n(P)$ *be a function that determines the number of facts derived during Standard Semi-Naive evaluation [2] of program P.* $\mathcal{S}n(P^{bits}) \leq \mathcal{S}n(P^{mg})$.

The proof of this theorem is straightforward by considering that now elimination of duplicated facts can be truly obtained.

To check whether $G \sqsupseteq S$ one must check whether the variables of S are bound by any of the ground parameters of G. Considering adornments as binary numbers one can implement this procedure through a simple logical operation on sequences of bits. Bearing in mind that '0' represents a position of free variable, performing an binary **or** operation over two adornment sequences yields another adornment sequence that represents the bound/free position in the parameters of both atoms after being unified. If a position is bound in one atom then after unification the same position is bound on both atoms. Recall that subsumption can be reduced to checking whether the m.g.u between G and S does not bind any variable in S. Thus, if the resultant adornment sequence of the logical **or** operation matches the sequence representative of S then $G \sqsupseteq S$.

Consider the following example; the atoms $\mathsf{p(a, Y, Z)}$ and $\mathsf{p(a, b, c)}$ have as m.g.u the substitution $\{\mathsf{Y/b, Z/c}\}$. The atoms after unification are both $\mathsf{p(a, b, c)}$ which corresponds to the adornment '111'. The first atom is represented by the adornment '100' and the second by '111'. Performing 100 **or** 111 yields 111. The first fact subsumes the second, because the resulting adornment sequence obtained from the **or** operation is equal to the sequence of the second fact. The fact that these two sequences are equal means that the m.g.u. does not perform any substitution on the variables of the second atom. Since Magic Sets are used, no aliasing of variables occurs (all terms with distinct variables). This justifies why binary numbers can be used to check subsumption.

We can summarize the subsumption algorithm in the following way. Consider that we want to check whether $G^\alpha \sqsupseteq S^\beta$, where α and β are the adornments. Then *subsumes* is defined as:

$$subsumes(G^\alpha, S^\beta) \leftrightarrow \mathbf{or}(\alpha, \beta, \beta) \ \& \ match(G, S).$$

When two magic facts succeed in the logical **or** test, one has to confirm whether the bound positions of both facts that coincide represent parameters that match. In other words, one has to perform pattern matching between the bound parameters of both facts. Since we are dealing with adorned magic facts, the parameters in these facts are all ground, corresponding to the bindings to be passed. In the definition described above this corresponds to the predicate *match*. However, this procedure *match* must be adjusted because we need to know the adornments to determine again which arguments in G correspond to which in S. Consider two magic facts $mag^{sp}(\vec{s})$ and $mag^{ge}(\vec{g})$, of which the first is more specific and the second more general. To perform pattern matching one compares the adornments ge and sp. From this comparison one matches only the positions on $\vec{s}$ and $\vec{g}$ that have '1' on both sp and ge, assuming that we already work with the translated sequences. As an example, consider the magic facts mag_p(0010, a) and mag_p(1011, j, a, c). Comparing the adornment sequences tells us that it is only necessary to match the third position in both facts. This is equivalent to comparing the first (and only) argument from the former (which is the constant a) with the second argument of the latter fact (constant a also).

We can also determine the positions to be compared through binary operations with the translated adornment sequences into binary numbers. Checking the bits in both sequences that are **on** i.e. assigned with 1, can be performed by successive operations of shifting and binary conjunctions. A variable is assigned with a binary number that has the same number of bits as the adornment sequences and all the bits turned off (i.e. 0) except the left-most one. Thus, for five bits the variable is assigned with '10000'. We assume that there is a pointer for each magic fact pointing to the list of parameters. Two binary conjunctions between the two adornment sequences and the variable are done. Matching between the pointed parameters is only performed if both conjunctions yield non-zero results. Now, for each conjunction that gives non zero result the respective pointer is incremented. Finally a one bit right shifting operation on the used variable is performed. This process is repeated while the pointer of the most general fact does not point to nil i.e. the list of parameters is not totally visited. This ensures that the number of comparisons between arguments of the two magic facts coincides with the number of parameters of the most general magic fact.

Let us consider an example with the magic facts mag_p(001, c) and mag_p(101, a, c). The auxiliary variable is assigned with '100'. Initially the pointer of the first magic fact points to the parameter c and the second to the parameter a. The conjunction '100 & 001 = 000' and '100 & 101 = 100' do not respect the first requirement. Thus no matching is performed and only the pointer for the second magic fact is incremented, pointing now to the constant c. Shifting the variable gives the binary number '010'. Both conjunctions yield zero as result. Therefore no matching is performed and no pointer is incremented. After the shifting, the variable has the value '001'. The operations are repeated and this time both conjunctions yield non zero results i.e. '001 & 001 = 001' and '001 & 101 = 001'. Thus, matching between pointed parameters is performed, which corresponds to apply matching between the constant c from the first magic fact with the constant c from the second magic fact.

4.2. The Algorithm

Finally, we are in position to present the complete *subsumes* algorithm. We use **or** and & to denote the binary operations of disjunction and conjunction, respectively. Two adorned magic facts, $mag\_p^{sp}(\vec{s})$ and $mag\_p^{ge}(\vec{g})$, participate in the algorithm. The original magic facts are translated into respectively $mag\_p(sp', \vec{s})$ and $mag\_p(ge', \vec{g})$. P_g is the pointer to the list of parameters in $\vec{g}$ and P_s is the pointer to the list of parameters in $\vec{s}$. Initially both point to the first argument of each magic atom. The algorithm checks whether $mag\_p^{ge}(\vec{g}) \sqsupseteq mag\_p^{sp}(\vec{s})$.

Algorithm *Subsumes*

```
1. if $sp' \neq sp'$ or $ge'$ then fail and exit.
2. else check pattern matching between $\vec{s}$ and $\vec{g}$.
   {The algorithm goes through $ge'$ and $sp'$, from left to right, to determine
   the positions to be matched.}
   $Aux := 1 << (n-1)$ where $n$ is the number of bits in the sequences $sp'$ and
   $ge'$. {shift to the left $n-1$ times the number 1 in binary format}
   Do while $P_g \neq nil$ {does not point to nil}
      $G := ge'$ & $Aux$;
      $S := sp'$ & $Aux$;
      if $G \neq 0$ and $S \neq 0$ then
         if not match$(P_g, P_s)$ then fail and exit;
      if $G \neq 0$ then make $P_g$ point to next position;
      if $S \neq 0$ then make $P_s$ point to next position;
      $Aux >> 1$ {shift once to the right};
   Endwhile
3. succeed.
```

The first step of the algorithm works as a preliminary test. The second step performs pattern matching. Note that the algorithm stops when all the arguments of the more general atom are visited ($\vec{g}$).

Let us consider some examples in the application of subsumption to the elimination of redundant magic facts derived during semi-naive evaluation: The calls ?p(a, Y, Z) and ?p(a, b, Z) correspond to the magic facts $\mathsf{mag\_p}^{\mathsf{bff}}(\mathsf{a})$ and $\mathsf{mag\_p}^{\mathsf{bbf}}(\mathsf{a}, \mathsf{b})$, respectively. Suppose the former is a previously derived fact and the latter is a new fact. We want to check whether $\mathsf{mag\_p}^{\mathsf{bff}}(\mathsf{a}) \sqsupseteq \mathsf{mag\_p}^{\mathsf{bbf}}(\mathsf{a}, \mathsf{b})$. Performing '100 or 110' results in '110' which is equal to the sequence in the new fact. Next, both sequences of arguments match since the first binding of the first fact (a) matches the first binding of the second (a). Therefore $\mathsf{mag\_p}^{\mathsf{bff}}(\mathsf{a}) \sqsupseteq \mathsf{mag\_p}^{\mathsf{bbf}}(\mathsf{a}, \mathsf{b})$. In a semi-naive evaluation the new fact would be eliminated, meaning that redundant computation associated with this fact would be avoided.

Consider the case where neither of the atoms subsumes the other. For instance the queries ?p(a, Y) and ?p(X, a), are represented by the magic facts $\mathsf{mag\_p}^{\mathsf{bf}}(\mathsf{a})$ and $\mathsf{mag\_p}^{\mathsf{fb}}(\mathsf{a})$. The operation '10 or 01' gives '11' as result. Thus, the algorithm returns failure. Consider finally an example with different bindings. Assume the magic facts $\mathsf{mag\_p}^{\mathsf{fbf}}(\mathsf{c})$ and $\mathsf{mag\_p}^{\mathsf{fbb}}(\mathsf{a}, \mathsf{c})$. The adornments checking succeeds since 010 or 011 = 011. However comparing the bindings gives failure because $\mathsf{c} \neq \mathsf{a}$.

The algorithm complexity is characterized by a $\mathcal{O}(m)$ behaviour where m is the number of arguments of the more general magic atom i.e. $m = length(\vec{g})$. Here, m also represents the number of comparisons performed during pattern matching i.e. the second step of the subsumption algorithm. The logical operations over adornments are negligible because they can be implemented at a machine register level. Proofs of soundness and completeness of the algorithm can be found in [1].

In practical terms the problem that one has to address is how to efficiently perform subsumption between one newly derived adorned magic fact and a *set* of previously derived adorned magic facts. Thus, we have to extend the proposed algorithm to include a proper mechanism for the indexing of derived adorned magic facts. In [13] a *trie*-like structure was proposed to index calls and their computed answers in a tabulated top-down procedure (XSB Prolog). Given a fixed order of term traversal, tries can be used to index terms (in our case magic facts). The major advantages of these structures is that it gives a collapsed check/insert operation. In our case, performing subsumption requires one traversal for each binary sequence in the trie that satisfies step 1 of our algorithm. Insertion is collapsed with one of these traversals performed during subsumption checking. The traversal is the one where failure occurs during step 2 of our algorithm and where the adornments sequences coincide. When traversing the trie, the described bit operations of our algorithm are executed: Step 1 of the subsumption algorithm is performed according to the first parameter of each term (which is the adornment sequence). For the terms where this step succeeds the remaining path is traversed according to the bits operations described in step 2.

5. Examples

We take the previous ancestor example of section 4 for demonstrating the benefits of the proposed adornment process and the new subsumption checking algorithm. The example will be executed by semi-naive evaluation incorporating the new subsumption checking to determine whether newly generated magic facts should be eliminated. These two proposals overcome the redundancy in the evaluation observed in section 4. Applying the rewriting of definition 5 to this example yields:

$\mathsf{anc(X, Y)} \leftarrow \mathsf{par(X, Y)}\ \&\ \mathsf{mag\_anc(01, Y)}.$
$\mathsf{anc(X, Y)} \leftarrow \mathsf{par(X, Z)}\ \&\ \mathsf{anc(Z, Y)}\ \&\ \mathsf{mag\_anc(01, Y)}.$
$\mathsf{mag\_anc(11, Z, Y)} \leftarrow \mathsf{par(X, Z)}\ \&\ \mathsf{mag\_anc(01, Y)}.$

$\mathsf{anc(X, Y)} \leftarrow \mathsf{par(X, Y)}\ \&\ \mathsf{mag\_anc(11, X, Y)}.$
$\mathsf{anc(X, Y)} \leftarrow \mathsf{par(X, Z)}\ \&\ \mathsf{anc(Z, Y)}\ \&\ \mathsf{mag\_anc(11, X, Y)}.$
$\mathsf{mag\_anc(11, Z, Y)} \leftarrow \mathsf{par(X, Z)}\ \&\ \mathsf{mag\_anc(11, X, Y)}.$

Semi-naive evaluation, which includes our subsumption checking algorithm, for the same query is:

$$\mathsf{T}^1 = \mathsf{EDB} \cup \{\mathsf{mag\_anc(01, d)}\}$$

$$\mathsf{T}^2_{\mathsf{before}} = \mathsf{T}^1 \cup \{\mathsf{anc(c, d)}, \mathsf{mag\_anc(11, b, d)}, \mathsf{mag\_anc(11, c, d)}, \mathsf{mag\_anc(11, d, d)}\}$$

$$\mathsf{T}^2_{\mathsf{after}} = \mathsf{T}^1 \cup \{\mathsf{anc(c, d)}\}$$

$\mathsf{T}^3 = \mathsf{T}^2 \cup \{\mathsf{anc(b,d)}\}$

$\mathsf{T}^4 = \mathsf{T}^3 \cup \{\mathsf{anc(a,d)}\}$

We split the relevant steps of the semi-naive evaluation into $\mathsf{T}_{\mathsf{before}}$ and $\mathsf{T}_{\mathsf{after}}$, meaning respectively the facts derived before and preserved after subsumption checking is applied.

In the evaluation of the second version of the program only the first two rules are fired. Redundant computation of step 2 in the evaluation of the first version of this example is eliminated because the redundant magic facts are subsumed by the initial query. One consequence of this checking and of the way adornments are performed is that derivation of duplicate facts for anc is eliminated. This derivation of duplicates appears in the semi-naive evaluation of the first version of this program (section 4), on steps 3 and 4.
Consider the same example but now for a cyclic graph which is represented by the EDB:

```
par(a, b).
par(b, c).
par(c, d).
par(d, e).
par(e, a).
```

Evaluation of the original magic rewriting of the same program for the query ?anc(X, e) is:

$\mathsf{T}^1 = \mathsf{EDB} \cup \{\mathsf{mag\_anc}^{\mathsf{fb}}(\mathsf{e})\}$

$\mathsf{T}^2 = \mathsf{T}^1 \cup \{\mathsf{anc}^{\mathsf{fb}}(\mathsf{d,e}), \mathsf{mag\_anc}^{\mathsf{bb}}(\mathsf{b,e}), \mathsf{mag\_anc}^{\mathsf{bb}}(\mathsf{c,e}), \mathsf{mag\_anc}^{\mathsf{bb}}(\mathsf{d,e}),$
$\quad \mathsf{mag\_anc}^{\mathsf{bb}}(\mathsf{e,e}), \mathsf{mag\_anc}^{\mathsf{bb}}(\mathsf{a,e})\}$

$\mathsf{T}^3 = \mathsf{T}^2 \cup \{\mathsf{anc}^{\mathsf{bb}}(\mathsf{d,e})\}$

$\mathsf{T}^4 = \mathsf{T}^3 \cup \{\mathsf{anc}^{\mathsf{bb}}(\mathsf{c,e}), \mathsf{anc}^{\mathsf{fb}}(\mathsf{c,e})\}$

$\mathsf{T}^5 = \mathsf{T}^4 \cup \{\mathsf{anc}^{\mathsf{bb}}(\mathsf{b,e}), \mathsf{anc}^{\mathsf{fb}}(\mathsf{b,e})\}$

$\mathsf{T}^6 = \mathsf{T}^5 \cup \{\mathsf{anc}^{\mathsf{bb}}(\mathsf{a,e}), \mathsf{anc}^{\mathsf{fb}}(\mathsf{a,e})\}$

$\mathsf{T}^7 = \mathsf{T}^6 \cup \{\mathsf{anc}^{\mathsf{bb}}(\mathsf{e,e}), \mathsf{anc}^{\mathsf{fb}}(\mathsf{e,e})\}$

Semi-naive evaluation with our subsumption checking algorithm for the same query is:

$\mathsf{T}^1 = \mathsf{EDB} \cup \{\mathsf{mag\_anc(01,e)}\}$

$\mathsf{T}^2_{\mathsf{before}} = \mathsf{T}^1 \cup \{\mathsf{anc(d,e)}, \mathsf{mag\_anc(11,b,e)}, \mathsf{mag\_anc(11,c,e)}, \mathsf{mag\_anc(11,d,e)},$
$\quad \mathsf{mag\_anc(11,e,e)}, \mathsf{mag\_anc(11,a,e)}\}$

$\mathsf{T}^2_{\mathsf{after}} = \mathsf{T}^1 \cup \{\mathsf{anc(d,e)}\}$

$\mathsf{T}^3 = \mathsf{T}^2 \cup \{\mathsf{anc(c,e)}\}$

$\mathsf{T}^4 = \mathsf{T}^3 \cup \{\mathsf{anc(b,e)}\}$

$\mathsf{T}^5 = \mathsf{T}^4 \cup \{\mathsf{anc(a,e)}\}$

$\mathsf{T}^6 = \mathsf{T}^5 \cup \{\mathsf{anc(e,e)}\}$

Again, in step 2, subsumption checking prevents the use of redundant magic facts, namely the ones with the $\mathsf{mag\_anc}^{\mathsf{bb}}$ adornment. Consequently and due to the way we apply adornments, the repeated answers for anc with the different adornments are not derived.

6. Discussion

In [14], it is shown that within Magic Sets the idea that more bound parameters in a query is always better than fewer is not correct. In other words, computing $?p(a,b)$ is not always better than computing the query $?p(a,Y)$ and checking whether b is in the answer. Sagiv shows that in some examples having the first query with the adornment p^{bb} leads to the appearance of the adornment p^{bf} in the body of the rules defining p. However, this implies the derivation of magic rules to the adornment p^{bf} and also to the adornment p^{bb}. Thus, recomputation will arise. Furthermore the same answers will be generated for the adornment p^{bb} and p^{bf}. This seems to be an evidence that our work and [14] address a similar problem. Sagiv proposes a new program transformation to factorize predicates into new ones that correspond to the bound and free arguments described in the adornments. We address the same problem by simply introducing a new subsumption checking algorithm with an adornment process that is only applied to magic literals.

It is generally accepted by the Deductive Databases community e.g. [14], that the number of derived facts in a computation is a good indication of the relative efficiency of the evaluation method. With the examples of the last section, we have shown that the efficiency of the bottom-up evaluation is improved. Our proposal can reduce evaluation from $\mathcal{O}(n^2)$ complexity to $\mathcal{O}(n)$, where n is the number of EDB facts (which is actually what happens in the presented examples of transitive closure), for non subsumption-free[2] [7] magic programs. Obviously, with subsumption-free programs our techniques perform poorly and worst than the standard combination of semi-naive evaluation and magic sets rewriting due to the burden of the new "semantic subsumption" of magic facts. Another important overhead is introduced by the removal of adornments from the literals in the bodies of rules. Without adornments no indexing of answers can be applied and consequently irrelevant facts can be tried in the bodies of rules.

Other techniques exist to improve standard magic sets as for instance *factoring* [8] and the proposal in [6]. In general, factoring a program is an undecidable problem and the application of the proposal in [6] is restricted to left- right- and multi-linear programs. Actually, factoring could not be applied to the ancestor example of section 5 with a fb query. However, it remains to be investigated what is the inter-relation between these proposals and ours.

With our proposal an efficient tabulation technique is obtained in bottom-up evaluation, since now the total reuse of previous computation occurs. Our bottom-

[2]Subsumption-free programs are defined in [7]. Here we assume magic rewritten programs according to definition 5 where subsumption is defined through the algorithm of section 4.2

up mechanism can be related to the OLDT proof procedure [17] but where no indexing of answers occurs. The proposed subsumption checking algorithm is equivalent to the *instance* checking included in the OLDT procedure.

7. Conclusions

In this paper we identified that the characteristic features of tabulation were not present in bottom-up with goal orientation. Namely, we observed that subsumption checking between subgoals (magic facts) was not implemented and sharing of derived facts between literals of the same predicate was not obtained. The desirable features of tabulation were rectified by proposing a new adornment process and an algorithm for checking subsumption over adorned magic facts. Clearly, performing subsumption checking carries additional costs. However, as previously shown, first in the literature for the case of subsumption checking in tabulated top down evaluation e.g. [20, 17], and here with the examples, these overheads are negligible when compared with redundant computation that can (possibly) be avoided. Furthermore, the proposed algorithm was shown to have a reasonable complexity which indicates that it is efficient enough to overcome the burden associated with subsumption checking.

The proposed algorithm should be implemented in a way that enables the switching on/off of the subsumption checking, before an evaluation is performed. This implementation policy follows, for instance, the way other optimization techniques appear in the deductive database system CORAL [12]. In this way, one could switch on in situations where different instances of the same magic fact are derived and switch off for subsumption-free programs.

Acknowledgements: The author would like to thank to anonymous referees for the valuable comments and suggestions. Thanks to Mario Florido, Alipio Jorge, Jorge Sousa Pinto and Ian Mackie for the comments given in earlier drafts of this paper. This work was partially supported by Junta Nacional de Investigação Científica e Tecnológica - Programa PRAXIS XXI grant number BD/2832/94.

REFERENCES

1. Azevedo P. J., *Magic Sets with Full Sharing*, Technical Report, Departamento de Informatica, Universidade do Minho 1995.

2. Balbin I., Ramamohanarao K., *A Generalization of the Differential Approach to Recursive Query Evaluation* in Journal of Logic Programming 1987, pp 259-262.

3. Bancilhon F., Maier D., Sagiv Y., Ullman J. *Magic Sets and other strange ways to Implement Logic Programs* in Proceedings of the 5th Symposium on Principles of Databases Systems, (PODS) 1986.

4. Beeri C., Ramakrishnan R. *On the Power of Magic* in Journal of Logic Programming vol 10, pp 255-299, 1991.

5. Chen W., Warren D.S., *Query Evaluation under the Well Founded Semantics* in Proceedings of the 12th Symposium on Principles of Database Systems, (PODS) 1993.

6. Kemp D., Ramamohanarao K., Somogyi Z., *Right-, left- and multi-linear rule transformation that maintain context information* in Proceedings of the 16th International Conference on Very Large Databases (VLDB), Australia 1990.

7. Maher M., Ramakrishnan R. *Deja Vu in Fixpoints of Logic Programs* in Proceedings of the North American Conference on Logic Programming, Cleveland, Ohio 1989.

8. Naughton J., Ramakrishnan R., Sagiv Y., Ullman *Argument Reduction Through Factoring* in Proceedings of the 15th International Conference on Very Large Databases (VLDB), Amsterdam 1989.

9. Pereira F., Shieber S., *Prolog and Natural Language Analysis* in Language & Information no. 10, Stanford, CA: Center for Study of Language and Information, 1987.

10. Ramakrishnan R., Beeri C., Krishnamurthy R., *Optimizing Existential Datalog Queries* in Proceedings of the 7th Symposium on Principles of Databases Systems (PODS), Austin 1988.

11. Ramakrishnan R., *Magic Templates: A Spellbinding Approach to Logic Programs* in Journal of Logic Programming pp 189-216, vol 11, 1991.

12. Ramakrishnan R., Srivastava D., Sudarshan S., *CORAL - Control, Relations and Logic* in Proceedings of the 18th Very Large DataBases Conference, Vancouver, Canada 1992.

13. Ramakrishnan I., Rao P., Swift T., Warren D.S., *Efficient Tabling Mechanisms for Logic Programs* in Proceedings of the Twelfth International Conference on Logic Programming, pp 697-711, Kanagawa, Japan, 1995.

14. Sagiv Y., *Is There Anything Better than Magic?* in Proceedings of the North American Conference on Logic programming, 1990.

15. Sagonas K., Swift T., Warren D.S., *XSB as an Efficient Deductive Database Engine* in Proceedings of SIGMOD 1994 Conference ACM.

16. Seki H. *On The Power of Alexander Templates* in Proceedings of the 8th Symposium on Principles of Databases Systems, (PODS) 1989.

17. Tamaki H., Sato T., *OLD Resolution with Tabulation* in Proceedings of the 3rd International Conference on Logic Programming, London U.K. 1986, pp 84-98.

18. Warren D. S., *Memoing for Logic Programs* in Communications of the ACM Vol 35, No 3, March 1992, pp 93-111.

19. Ullman J., *Bottom-up beats Top-down for Datalog* in Proceedings of the 8th Symposium on Principles of Databases Systems, (PODS) 1989.

20. Vieille L., *Recursive Query Processing: The Power of Logic* in Theoretical Computer Science, vol 69, Elsevier Science Publishing 1989, pp 1-53.