A Framework for Goal-Directed Query Evaluation with Negation

Stefan Brass

Martin-Luther-Universität Halle-Wittenberg, Institut für Informatik,
Von-Seckendorff-Platz 1, D-06099 Halle (Saale), Germany
brass@informatik.uni-halle.de

Abstract. This paper contains a proposal how goal-directed query evaluation for the well-founded semantics WFS (and other negation semantics) can be done based on elementary program transformations. It also gives a new look at the author's SLDMagic method, which has several advantages over the standard magic set method (e.g., for tail recursions).

1 Introduction

The efficient evaluation of queries to logic programs with nonmonotic negation remains an everlasting problem. Of course, big achievements have been made, but at the same time problem size and complexity grows, so that any further progress can increase the practical applicability of logic-based, declarative programming.

In this paper, we consider non-disjunctive Datalog, i.e. pure Prolog without function symbols, but with unrestricted negation in the body. All rules must be range-restricted (allowed), i.e. variables appearing in the head or a negative body literal must also appear in a positive body literal.

We are mainly interested in the well-founded semantics WFS, but since our method is based on elementary program transformations, it can also be used as a pre-computation step for other semantics. Of course, it is well-known that because of odd loops over negation, for the stable model semantics it does not suffice to follow only the predicate calls from a given goal. But see, e.g. [2, 11].

The magic set transformation [1] is the best known method for goal-directed query evaluation in deductive databases. However, it has a number of problems:

- For tail recursions, magic sets are significantly slower than SLD-resolution (e.g. a quadratic number of facts derived compared with an SLD-tree containing only a linear number of nodes/literals). This problem applies to all methods which store literals implicitly proven in the SLD-tree ("lemmas").
- It sometimes transforms non-recursive programs into recursive ones. In the same way, a stratified program can be transformed into a non-stratified one.
- It can only pass values for arguments to called predicates, not more general conditions. Furthermore, optimizations based on the evaluation sequence of called literals are restricted to the bodies of single rules.
- Quite often, variable bindings are projected away in order to call a predicate, and are later recovered by a costly join when the predicate succeeded.

While some of these problems have been solved with specific optimizations, our "SLDMagic"-method [3] solved all these problems by simulating SLD-resolution bottom-up.

Of course, sometimes magic sets are better: If the same literal is called repeatedly in different contexts, magic sets proves it only once, whereas SLD resolution proves it again each time. Therefore, SLDMagic had the possibility to mark positive body literals with $\mathsf{call}(\ldots)$, in which case they were proven separately in a different SLD-tree. I.e. we assume that the programmer or an automatic optimizer marks some of the positive body literals of the rules with the special keyword "call". In this way, the best of both methods can be combined. Furthermore, when we want to compute the structure of occurring goals already at "compile-time" (when the facts/relations for database predicates are not yet known), we needed to introduce at least one "call" in recursions which are not tail-recursions. Since call-literals are proven in a subproof (much like negation as failure), the length of the occurring goals in the SLD-tree became bounded, and we could encode entire goals as single facts (where the arguments represented the data values only known at "run time").

However, SLDMagic could not handle negation. It is the purpose of this paper to remedy this problem, and to give a different look at the method, which opens perspectives for further improvements. Whereas earlier, we described the method by means of partial evaluation of a meta-interpreter, we now combine it with our work on elementary program transformations.

Together with Jürgen Dix, we investigated ways to characterize nonmonotonic semantics of (disjunctive) logic programs by means of elementary program transformations [4, 5]. We used the notion of conditional facts, introduced by Bry [7] and Dung/Kanchansut [10]. A conditional fact is a ground rule with only negative body literals. If a nonmonotonic semantics permits certain simple transformations, in particular the "Generalized Principle of Partial Evaluation" [9, 12] (which is simply unfolding for non-disjunctive programs), and the elimination of tautologies, any (ground) program can be equivalently transformed into a set of conditional facts. If we also permit the evaluation of negative body literals in obvious cases, we can compute a unique normal form of the program, called the "residual program". From this, one can directly read the well-founded model: If A is given as (unconditional) fact, it is true, if there is no rule with A in the head, it is false, and in all other cases, it is undefined. The residual program is also equivalent to the original program under the stable model semantics.

Because the residual program can grow to exponential size in rare cases, we restricted unfolding and "delayed" also positive body literals until their truth value became obvious (as for the negative body literals before) [6]. Combined with magic sets, this resulted in a competitive evaluation procedure for WFS. But there is further optimization potential if we use ideas from SLDMagic. It is also nice if the framework is based entirely on elementary program transformations.

This paper contains only some preliminary ideas (it is a "short paper" about work in progress). A more complete version is being prepared. Progress will be reported at: `http://www.informatik.uni-halle.de/~brass/negeval/`.

There are some obvious similarities to SLG-resolution [8, 13]. However, when a tail-recursive predicate is tabled in order to ensure termination, SLG-resolution has the same problem as magic sets. Furthermore, our approach is explained based only on rules, whereas SLG-resolution needs more complex data structures. But much more work has been done on the efficient implementation of SLG-resolution in the XSB system, whereas our approach is still at the beginning.

2 Goal-Directed Query Evaluation Based on Program Transformations

We assume that the query is given as a rule $\mathsf{answer}(\mathsf{X}_1, \dots, \mathsf{X}_m) \leftarrow \mathsf{B}_1 \wedge \dots \wedge \mathsf{B}_n$, where the special predicate answer does not otherwise appear in the program. The body of this query (rule) is the classical query. In this way, we do not have to track substitutions for the variables from the query while the proof proceeds. This is automatically done if we compute the instances of the special predicate answer which are derivable from the program plus this rule.

The method generates rules which must be considered (starting with this query rule). The rule body is a goal, as would appear in an SLD-tree. The rule head is the literal which has to be proven. First this is $\mathsf{answer}(\mathsf{X}_1, \dots, \mathsf{X}_m)$, but as the proof progresses, the answer variables X_i are instantiated. Furthermore, negative body literals and call-literals cause subqueries to appear.

The occurring rules can be seen as being generated from the given program (extended by the query rule) using elementary transformations. However, in contrast to our earlier work, there are two important differences:

– The occurring rules are often not ground. In the theoretical part of our work on negation semantics, we started with the full ground instantiation of the given program. Of course, if we now want to use program transformations as a practical means of computation, we must work with non-ground rules. They can be understood as a compact representation of the set of their ground instances. Most modern semantics including WFS do not distinguish between a rule with variables and its set of ground instances.
– For goal-directed query evaluation, we do not want to consider the entire program. The "relevance" property of a nonmonotonic semantics [9] ensures that it is sufficient to look only at ground literals which are reachable from the query via the call-graph. WFS has the relevance property, the stable model semantics does not, but see [11]. Note that relevance is applied repeatedly during the evaluation. When we found that a rule instance is not applicable, it vanishes from the call graph and might remove large parts of the program.

Of course, we do not first take the entire program, and then delete non-relevant parts. Instead, we have a "working set" of rules we consider. We apply transformations on this set until a normal form is reached, i.e. until no further transformation is applicable. We only have to ensure that as long as there still is a relevant rule instance in the given program, it is considered, i.e. a transformation remains applicable.

In order to improve termination, we need to avoid the generation of rules which differ from an already generated rule only by a renaming of variables.

Definition 1 (Variable Normalization). *Let an infinite sequence* $\mathsf{V}_1, \mathsf{V}_2, \ldots$ *of variables be given. A rule* $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$ *is variable-normalized iff it only contains variables from the set* $\{\mathsf{V}_1, \mathsf{V}_2, \ldots\}$ *and for each occurrence of* V_i*, all variables* $\mathsf{V}_1, \ldots, \mathsf{V}_{i-1}$ *occur to the left. The function* std *renames the variables of a rule to* $\mathsf{V}_1, \mathsf{V}_2, \ldots$ *in the order of occurrence, i.e. produces a variant of the given rule which is variable-normalized.*

Our method works with a set of rules which is initialized with the query. We write R for the current rule set to distinguish it from the given logic program P. We use a second set D of rules for the "deleted" rules. In this way, both sets R and D can only grow monotonically during the computation, which improves termination: It is not possible that a rule is added, deleted, and then added again. However, only the non-deleted rules in R really participate in the computation.

Definition 2 (Computation State). *A computation state is a pair* (R, D) *of sets of variable-normalized rules such that* $\mathsf{D} \subseteq \mathsf{R}$*. A rule in* $\mathsf{R} - \mathsf{D}$ *is called active, a rule in* D *is called deleted.*

Let the query Q *be* $\mathsf{answer}(\mathsf{X}_1, \ldots, \mathsf{X}_m) \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$*. The initial computation state is* $(\mathsf{R}_0, \mathsf{D}_0)$ *with* $\mathsf{R}_0 := \{\mathsf{std}(\mathsf{Q})\}$ *and* $\mathsf{D}_0 := \emptyset$*.*

2.1 Positive Body Literals

In the following, we use the term "positive body literal" for a body literal without negation and without "call". Literals with "call" are "call-literals" (they are never negated because negation already implies a subproof like "call" does).

Positive body literals are solved with unfolding (an SLD resolution step). If a rule contains several positive body literals, a selection function restricts unfolding to one of these. We require that a recursive positive body literal can only be selected last (when there are no other positive body literals). This implies that there can be only one recursive positive body literal, but this is no restriction: Other such literals can be made call-literals. The purpose of this condition is to make the length of the occurring rules bounded (to ensure termination). Negative body literals which are added during the tail-recursion are no problem if we eliminate duplicates: They are ground, so the total number is still bounded (although the bound depends on the data, whereas the bound for the positive body literals depends only on the program rules). Non-ground call-literals which are added during the repeated unfolding of a tail-recursive rule cannot be permitted. But again, this is no restriction because we can make the tail-recursive literal a call-literal, too (at the cost of losing the tail-recursion optimization).

The selection function only restricts the unfolding of positive body literals. Work on negative body literals and call-literals is not restricted, although an implementation is free to decide in each step which of several applicable transformations it uses. The reason why we cannot prescribe a single "active" literal in each rule is that because of rules like $\mathsf{p} \leftarrow \neg\mathsf{p}$, the evaluation of a negative body literal (in this case, $\neg\mathsf{p}$) can "block". But there might be another body literal which would fail (this is similar to the "fairness" requirement of SLD resolution). The same can happen with call-literals.

The computation steps are relations between computation states. An implementation can follow any path until no further transformation is possible.

Definition 3 (Unfolding). *Let* $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$ *be a rule in* $\mathsf{R} - \mathsf{D}$*, where* B_i *is a positive literal selected by the selection function. Let further* $\mathsf{A}' \leftarrow \mathsf{B}'_1 \wedge \cdots \wedge \mathsf{B}'_m$ *be a variant of a rule in* P *with fresh variables (i.e. the variables renamed such that they are disjoint from those occurring in* $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$*), such that* B_i *and* A' *are unifiable with most general unifier* θ*. Let*

$$\mathsf{R}' := \mathsf{R} \cup \{\mathsf{std}(\theta(\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_{i-1} \wedge \mathsf{B}'_1 \wedge \cdots \wedge \mathsf{B}'_m \wedge \mathsf{B}_{i+1} \wedge \cdots \wedge \mathsf{B}_n))\}$$

If $\mathsf{R}' \neq \mathsf{R}$*, we write* $(\mathsf{R}, \mathsf{D}) \mapsto_U (\mathsf{R}', \mathsf{D})$*.*

Note that because R is a set, unfolding with a rule like $\mathsf{p}(\mathsf{X}) \leftarrow \mathsf{p}(\mathsf{X})$ does not result in a new rule and therefore cannot lead to non-termination. We need to assume that the negation semantics permits the deletion of tautologies.

Definition 4 (Deletion After Complete Unfolding). *Let* $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$ *be a rule in* $\mathsf{R} - \mathsf{D}$*, where* B_i *is a positive literal selected by the selection function. Let all rules which can be generated from the given rule by unfolding be already contained in* R*, and let* $\mathsf{D}' := \mathsf{D} \cup \{\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n\}$*. If* $\mathsf{D}' \neq \mathsf{D}$*, the transformation step* $(\mathsf{R}, \mathsf{D}) \mapsto_D (\mathsf{R}, \mathsf{D}')$ *is permitted.*

If there is only a single matching rule, or one immediately unfolds with all matching rules (e.g. in case of set-oriented evaluation with a database predicate), one can "delete" the rule with the unfolded call immediately. But by separating the two steps, other evaluation orders are possible, e.g. doing a depth-first search.

2.2 Negative Body Literals

Definition 5 (Complement Call). *Let* $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$ *be a rule in* $\mathsf{R} - \mathsf{D}$*, and* B_i *be a negative ground literal. Let* A' *be the corresponding positive literal, i.e.* $\mathsf{B}_i = \neg\mathsf{A}'$*. Let* $\mathsf{R}' := \mathsf{R} \cup \{\mathsf{A}' \leftarrow \mathsf{A}'\}$*. If* $\mathsf{R}' \neq \mathsf{R}$*, we write* $(\mathsf{R}, \mathsf{D}) \mapsto_C (\mathsf{R}', \mathsf{D})$*.*

Of course, $\mathsf{A}' \leftarrow \mathsf{A}'$ is a tautology. But it is important because it sets up a new query. So when we want to work on a negative literal, we try to prove the corresponding positive literal. This is the same as SLDNF-resolution would do.

The next transformation handles the case where a negative literal is proven by failure to prove the corresponding positive literal.

Definition 6 (Positive Reduction). *Let* $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$ *be a rule in* $\mathsf{R} - \mathsf{D}$*, where* B_i *is a negative ground literal. Let* A' *be the corresponding positive literal,* *i.e.* $\mathsf{B}_i = \neg\mathsf{A}'$. *If* R *contains* $\mathsf{A}' \leftarrow \mathsf{A}'$, *but* $\mathsf{R} - \mathsf{D}$ *does not contain any rule with head* A', *then* $(\mathsf{R}, \mathsf{D}) \mapsto_P (\mathsf{R}', \mathsf{D}')$ *with*

$$\begin{aligned}\mathsf{R}' &:= \mathsf{R} \cup \{\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_{i-1} \wedge \mathsf{B}_{i+1} \wedge \cdots \wedge \mathsf{B}_n\}\\ \mathsf{D}' &:= \mathsf{D} \cup \{\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n\}.\end{aligned}$$

(The new rule is variable-normalized since B_i *was ground.)*

The next transformation handles the case that the selected negative body literal is obviously false, because the corresponding positive literal was proven:

Definition 7 (Negative Reduction). *Let* $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$ *be a rule in* $\mathsf{R} - \mathsf{D}$, *where* B_i *is a negative ground literal. Let* A' *be the corresponding positive literal, i.e.* $\mathsf{B}_i = \neg\mathsf{A}'$. *If* R *contains* A' *(as a rule with empty body, i.e. a fact), then* $(\mathsf{R}, \mathsf{D}) \mapsto_N (\mathsf{R}, \mathsf{D}')$ *with* $\mathsf{D}' := \mathsf{D} \cup \{\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n\}$.

2.3 Call Literals

The specially marked call-literals are semantically positive literals, but they are not solved by unfolding. Instead, a subproof is set up (as for negative literals):

Definition 8 (Start of Subproof). *Let* $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$ *be a rule in* $\mathsf{R} - \mathsf{D}$, *and the literal* B_i *be of the form* $\mathsf{call}(\mathsf{A}')$. *Let* $\mathsf{R}' := \mathsf{R} \cup \{\mathsf{A}' \leftarrow \mathsf{A}'\}$. *If* $\mathsf{R}' \neq \mathsf{R}$, *we write* $(\mathsf{R}, \mathsf{D}) \mapsto_S (\mathsf{R}', \mathsf{D})$.

The following transformation is similar to positive reduction combined with a very special case of unfolding:

Definition 9 (Return). *Let* $\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_n$ *be a rule in* $\mathsf{R} - \mathsf{D}$, *and the literal* B_i *be of the form* $\mathsf{call}(\mathsf{A}')$. *Suppose further that there is a rule* $\mathsf{A}'' \leftarrow \mathsf{B}'_1 \wedge \cdots \wedge \mathsf{B}'_m$ *where all body literals are negative (i.e. a conditional fact), such that* A' *and* A'' *are unifiable with most general unifier* θ. *Let*

$$\mathsf{R}' := \mathsf{R} \cup \{\mathsf{std}(\theta(\mathsf{A} \leftarrow \mathsf{B}_1 \wedge \cdots \wedge \mathsf{B}_{i-1} \wedge \mathsf{B}'_1 \wedge \cdots \wedge \mathsf{B}'_m \wedge \mathsf{B}_{i+1} \wedge \cdots \wedge \mathsf{B}_n))\}$$

If $\mathsf{R}' \neq \mathsf{R}$, *we write* $(\mathsf{R}, \mathsf{D}) \mapsto_R (\mathsf{R}', \mathsf{D})$.

A call is complete when a kind of small fixpoint is reached in the larger set of rules constructed. Negative literals can be evaluated later, but for positive body literals and call-literals, all possible derivations must be done:

Definition 10 (End of Subproof). *Let* $\mathsf{R}_0 \subseteq \mathsf{R}_1 \subseteq \mathsf{R}$ *be rule sets, such that*

- *Each rule in* R_0 *has a call-literal as selected literal,*
- *the transformations* $\mapsto_U$ *(Unfolding),* $\mapsto_S$ *(Start of Subproof),* $\mapsto_R$ *(Return) are not applicable in* R_1 *(i.e. everything derivable is already contained in* R_1*).*

If $\mathsf{R}_0 \not\subseteq \mathsf{D}$, *the "End of Subproof" transformation is applicable:*

$$(\mathsf{R}, \mathsf{D}) \mapsto_E (\mathsf{R}, \mathsf{D} \cup \mathsf{R}_0).$$

This transformation is relatively complicated because it includes a kind of loop detection. It must be able to handle cases like:

$$\mathsf{p(X)} \leftarrow \mathsf{call(q(X))}.$$
$$\mathsf{q(X)} \leftarrow \mathsf{call(p(X))}.$$

An alternative for the "return" operation is not to unfold, but just ground the call if it matches the head of an "extended conditional fact", which is a ground rule with only negative and call-literals in the body. The call-literal would be removed only if it matches a fact (without condition). This operation "success" of [6] (and the converse "failure") help to avoid a possible exponential blowup.

References

1. Beeri, C., Ramakrishnan, R.: On the power of magic. In: Proc. Sixth ACM Symp. on Principles of Database Systems (PODS'87). pp. 269–284. ACM (1987)
2. Bonatti, P.A., Pontelli, E., Son, T.C.: Credulous resolution for answer set programming. In: Proc. of the 23rd National Conf. on Artificial Intelligence (AAAI'08) - Volume 1. pp. 418–423. AAAI Press (2008)
3. Brass, S.: SLDMagic — the real magic (with applications to web queries). In: Lloyd, W., et al. (eds.) First International Conference on Computational Logic (CL'2000/DOOD'2000). pp. 1063–1077. No. 1861 in LNCS, Springer (2000)
4. Brass, S., Dix, J.: Characterizations of the stable semantics by partial evaluation. In: Nerode, A. (ed.) Logic Programming and Nonmonotonic Reasoning, Proc. of the Third Int. Conf. (LPNMR'95). pp. 85–98. No. 928 in LNCS, Springer (1995)
5. Brass, S., Dix, J.: A general approach to bottom-up computation of disjunctive semantics. In: Dix, J., Pereira, L.M., Przymusinski, T.C. (eds.) Nonmonotonic Extensions of Logic Programming, pp. 127–155. No. 927 in LNAI, Springer (1995)
6. Brass, S., Dix, J., Freitag, B., Zukowski, U.: Transformation-based bottom-up computation of the well-founded model. Theory and Practice of Logic Programming 1(5), 497–538 (2001)
7. Bry, F.: Logic programming as constructivism: A formalization and its application to databases. In: Proc. of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'89). pp. 34–50 (1989)
8. Chen, W., Warren, D.S.: A goal-oriented approach to computing the well-founded semantics. The Journal of Logic Programming 17, 279–300 (1993)
9. Dix, J., Müller, M.: Partial evaluation and relevance for approximations of the stable semantics. In: Ras, Z., Zemankova, M. (eds.) Proc. of the 8th Int. Symp. on Methodologies for Intelligent Systems. LNAI, Springer (1994)
10. Dung, P.M., Kanchansut, K.: A fixpoint approach to declarative semantics of logic programs. In: Proc. North American Conference on Logic Programming (NACLP'89). pp. 604–625 (1989)
11. Marple, K., Gupta, G.: Galliwasp: A goal-directed answer set solver. In: Albert, E. (ed.) Logic-Based Program Synthesis and Transformation. LNCS, vol. 7844, pp. 122–136. Springer Berlin Heidelberg (2013)
12. Sakama, C., Seki, H.: Partial deduction of disjunctive logic programs: A declarative approach. In: Fourth Int. Workshop on Logic Program Synthesis and Transformation (LOPSTR'94). LNCS, Springer (1994)
13. Swift, T.: Tabling for non-monotonic programming. Annals of Mathematics and Artificial Intelligence 25, 201–240 (1999)