Computing the Well-founded Semantics for Constraint Extensions of Datalog$^\neg$

David Toman

Department of Computer Science, University of Toronto
Toronto, Ontario, Canada M5S 1A4
david@cs.toronto.edu

Abstract. We present a new technique for computing the well-founded semantics for constraint extensions of Datalog$^\neg$. The method is based on tabulated resolution enhanced with a new refinement strategy for deriving negative conclusions. This approach leads to an efficient and terminating query evaluation algorithm that preserves the goal-oriented nature of the resolution based methods.

1 Introduction

The well-founded semantics of Datalog$^\neg$ programs [29, 30] provides a robust model for handling negation in deductive databases (and in general logic programming systems). There have been numerous proposals for query evaluation procedures under the well-founded semantics. The two main approaches to computing the well-founded semantics of Datalog$^\neg$ programs are the bottom-up methods, usually based on the *Alternating fixpoint* [10, 13, 20, 31], or the top-down methods, based on a variation of SLD-resolution [3, 4, 5, 19].

However, in all the cases, the query evaluation algorithms assume that the derived answers are *ground tuples*: the fixpoint method requires *allowed* Datalog$^\neg$ programs—a syntactically restricted class of queries for which all derivations can be safely performed on ground tuples; the resolution-based methods require all the negative subgoals to be ground and fail with an error if a non-ground negative literal is encountered during the evaluation.

The restriction to ground tuples is not natural (and often not feasible) when constraints are used as the data representation [9]: in this setting the sets of ground tuples represented by constraints are often infinite. On the other hand, the constraint representation has often better *closure properties* (cf. Section 2, Example 2.2 or [9]) than the ground representation.

In this paper we propose a top-down query evaluation procedure based on tabulated resolution enhanced with a new refinement phase—*Constraint Memoing(WF)*. The procedure computes answers for arbitrary constraint Datalog$^\neg$ (Datalog$^{\mathcal{C},\neg}$) queries under the well-founded semantics. The features of the proposed method are as follows:

- The query evaluation algorithm is independent of the chosen constraint class. It is parametrized by abstract operations associated with the constraint representation specific to the given constraint class. Thus the algorithm can be easily extended to accommodate various classes of constraints [9, 17, 21, 26].

- The termination of the algorithm is guaranteed. We provide a sufficient criterion (as a property of the given class of constraints) that guarantees termination of arbitrary Datalog$^{\mathcal{C},\neg}$ queries.
- The evaluation is goal-oriented. We show that our new method is never worse than the bottom-up method based on a constraint version of the alternating fixpoint. This remains true even for Datalog$^{\mathcal{C},\neg}$ programs transformed using MST [13].
- The method avoids recomputation of already known conclusions using a new *refinement* technique. Note that the bottom-up algorithms can not avoid the recomputation due to the nature of the alternating fixpoint (cf. Section 2.3),

Moreover, the proposed method can reuse results from the vast recent developments in the area of top-down evaluation of deductive queries [5, 4, 27], especially from the very efficient compilation techniques developed for the top-down query processing[1]. Results in [27] suggest that (for definite programs) most of these results can be used for query evaluation in the constraint case provided the groundness restrictions are lifted. Note also, that in the constraint setting the difference between set-at-a-time and tuple-at-a-time approaches is blurred[2]. Thus our method for computing the well-founded semantics can be used in both the tuple-at-a-time (SLG-like) and the set-at-a-time (QSQ-like) modes and directly benefit from the development of both the bottom-up and top-down technologies.

The rest of the paper is organized as follows: Section 2 introduces the necessary definitions of constraint class and of the well-founded semantics for (ground) Datalog with negation. Section 3 describes the proposed top-down evaluation method and shows its correctness. Section 4 discusses the implementation techniques applicable to the proposed method. Section 5 concludes the paper with a brief discussion of related work and possibilities of further improvements.

2 Preliminaries

In this section we introduce the two main concepts needed to define Constraint Memoing(WF) and to show its correctness. First we define a *constraint class*—an abstract representation of constraints together with operations on which the actual query evaluation procedure is parametrized. Then we present a standard definition of the well-founded semantics (for ground Datalog$^{\neg}$ programs) and show correctness and termination of the constraint version of an alternating fixpoint query evaluation procedure for Datalog$^{\mathcal{C},\neg}$.

[1] Benchmarks for main-memory systems can be found in [25, 11, 27] and for a disk-based system in [6].

[2] A set-at-a-time like computation is easily achieved by allowing finite disjunction constraints.

2.1 The Constraint Representation

We use constraints to encode possibly infinite sets of ground tuples. This is the main difference from the classical approach taken in virtually all proposals dealing with evaluation of deductive queries under the well-founded semantics. The abstract properties of the constraints are as follows:

Definition 2.1 (Constraint Class) *A* Constraint class *is a least set of constraints $\mathcal{C}$ containing $\mathcal{C}_0 \cup \{\text{true}\}$, is closed under conjunction, quantifier elimination, negation, and renaming, that is equipped with the following (computable) operations:*

Constraint Conjunction $\wedge^{\mathcal{C}} : \mathcal{C} \times \mathcal{C} \to \mathcal{C} \cup \{\bot\}$ *that for every pair of constraints C_1, C_2 computes the* conjunction $C_1 \wedge C_2$ *if the conjunction is satisfiable; otherwise returns $\bot$.*

Constraint Projection $\exists^{\mathcal{C}} : V \times \mathcal{C} \to 2^{\mathcal{C}}$ *that for every constraint C and every set $V \subset FV(C)$*[3] *computes the set $\{C_i\}$ (a* projection *of C to V) that satisfies the condition*

$$\left(\bigvee_{C_i \in \exists^{\mathcal{C}}(V,C)} C_i\right) \equiv \exists x_1. \cdots \exists x_l.C$$

where $x_j \in FV(C) - V$ for $0 < j \leq l$. The function is well defined and always returns a finite *subset of $\mathcal{C}$.*

Constraint Complementation $\neg^{\mathcal{C}} : \mathcal{C} \to 2^{\mathcal{C}}$ *that for every constraint C computes the set $\{C_i\}$ (a* complement *of C) that satisfies the condition*

$$\left(\bigvee_{C_i \in \neg^{\mathcal{C}}(C)} C_i\right) \equiv \neg C$$

Constraint Subsumption $\leq^{\mathcal{C}} : \mathcal{C} \times \mathcal{C} \to \text{bool}$ *that satisfies the condition*

$$C_1 \leq^{\mathcal{C}} C_2 \textit{ implies } C_1 \supset C_2$$

The elements of $\mathcal{C}$ are used as a finite representation of (possibly infinite) relations in a constraint database. The query evaluation over such representation is based on the operations defined on the elements of the constraint class: The first three operations are the equivalents of relational algebra join, projection, and complementation[4]. The last operation, the constraint subsumption, is used for duplicate elimination and is the key to proving termination the of query evaluation algorithm. Note that the relation induced by $\leq^{\mathcal{C}}$ can be *weaker* than the $\supset$ relation. However, a *better approximation* of $\supset$ relation by the $\leq^{\mathcal{C}}$ operation reduces the number of possible duplicate answers and improves the efficiency of the evaluation method. In the following text we omit the superscripts $^{\mathcal{C}}$. We also use a *strict*[5] version of the $\wedge^{\mathcal{C}}$ operation.

[3] $FV(C)$ is the set of free variables in C.

[4] This is possible due to the closure properties of the constraint class. We discuss various restricted versions of the complementation operation in Section 4.

[5] I.e., $\bot$-preserving.

Example 2.2 (Dense Order Constraints) Let $(\mathbf{Q}, <)$ be a linearly ordered set of rational numbers and $A \subset \mathbf{Q}$ a finite set. Then the carrier of the class of *Dense Order Constraints* is generated from the set $\mathcal{C}_0 = \{p < x, x < y, x < r : p, r \in A\}$. The constraint operations on this set are defined in, e.g., [8, 9, 12].

The following definition states a fundamental property of constraint classes on which the termination proofs of bottom-up query evaluation procedures are (implicitly) based [17, 21, 26, 28]:

Definition 2.3 (Constraint compact Constraint Class) *Let $\mathcal{C}$ be a constraint class. We call $\mathcal{C}$* constraint compact *if for every finite set of variables V and for every set $\mathcal{D} \subseteq \mathcal{C}$ such that $\forall C \in \mathcal{D}.FV(C) \subseteq V$ there is a finite subset $\mathcal{D}_{fin} \subseteq \mathcal{D}$ such that $\forall C \in \mathcal{D}.\exists C' \in \mathcal{D}_{fin}.C \leq^{\mathcal{C}} C'$. (i.e., $\mathcal{D}_{fin}$ covers $\mathcal{D}$ with respect to $\leq^{\mathcal{C}}$).*

This property plays a central role in the termination proof of the bottom-up alternating fixpoint based evaluation algorithms as well as the top-down query evaluation procedure developed in Section 3.

2.2 The Well-founded Semantics

In this section we review the basic definition of the well-founded semantics and the intuition behind the various query evaluation methods proposed for it.

Definition 2.4 (Datalog$^{\mathcal{C},\neg}$ Program) *Let $\mathcal{C}$ be a constraint class. A* atom *is a predicate symbol with distinct variables as its arguments. A* Datalog$^{\mathcal{C},\neg}$ program *is a set of clauses of the form $Q \leftarrow D, P_1, \ldots, P_k, \neg R_1, \ldots, \neg R_l$ where Q, P_i, and R_i are atoms and $D \in \mathcal{C}$.*

We can assume that the extensional database (EDB) is represented by a set of unit clauses of the form $A \leftarrow D$ as a part of the constraint program. A query over such database is a pair containing an atom and a constraint the returned tuples have to satisfy.

Definition 2.5 (Ground Datalog Program) *Let P be a* Datalog$^{\mathcal{C},\neg}$ *program. We define a set*

$$\begin{aligned}\text{ground}(P) = \{\ & Q\theta \leftarrow P_1\theta, \ldots, P_k\theta, \neg R_1\theta, \ldots, \neg R_l\theta : \\ & Q \leftarrow D, P_1, \ldots, P_k, \neg R_1, \ldots, \neg R_l \in P,\ \theta \models D\ \}\end{aligned}$$

where θ is a ground substitution[6]*.*

The set ground(P) is possibly infinite. However, the clauses in ground(P) do not contain constraints. Thus we can determine its *Well-founded model* using the standard definition [29]:

Definition 2.6 (Well-founded Model) *Let P be* Datalog$^{\mathcal{C},\neg}$ *program and I^+ and I^- disjoint sets of ground atoms. We call the pair (I^+, I^-)* a (ground) three-valued interpretation*.*

[6] A mapping from the variable names to the universe of the constraint class $\mathcal{C}$.

An immediate consequence of (I^+, I^-) with respect to P *is the set*

$$T_P(I^+, I^-) = \{Q : Q \leftarrow P_1, \ldots, , \neg R_l \in \text{ground}(P), P_i \in I^+, R_j \in I^-\}.$$

We call the mapping $T_P : (I^+, I^-) \mapsto T_P(I^+, I^-)$ *the* immediate consequence operator*.*

A set of ground atoms U *is* an unfounded set with respect to (I^+, I^-) and P *if for each* $Q \in U$ *and each clause* $Q \leftarrow P_1, \ldots, P_k, \neg R_1, \ldots, \neg R_l \in \text{ground}(P)$ *there is a* $P_i \in I^- \cup U$ *or a* $R_j \in I^+$*. The greatest unfounded set* $U_P(I^+, I^-)$ *with respect to* P *and* (I^+, I^-) *is the union of all the unfounded sets. We denote* U_P *the* greatest unfounded set operator *mapping* (I^+, I^-) *to* $U_P(I^+, I^-)$*.*

Let T_P *be the immediate consequence operator and* U_P *the greatest unfounded set operator. Then we define* $W_P(I^+, I^-) = (T_P(I^+, I^-), U_P(I^+, I^-))$ *and*

$$M_{\text{WF}}(P) = \text{lfp}\, \lambda(x, y).W_P(x, y)$$

to be the well-founded model of P*.*

W_P is monotone [31] and thus a unique $M_{\text{WF}}(P)$ exists for any P. Note that replacing T_P by T_P^k for $k \leq \omega$ in the definition of W_P leads to the same well founded model [19].

2.3 Closed-form Bottom-up Evaluation

Definition 2.6 does not give a constructive definition of the well-founded model. The problem is in the definition of the greatest unfounded set which is not constructive. However, the problem can be avoided by computing the complement of that set—the *least set of overestimates* consistent with the given program and interpretation. This approach is based on a well-known equality (stated with slight abuse of notation as) "$\text{gfp}\, X.F(X) = \neg\, \text{lfp}\, X.\neg F(\neg X)$", and can be traced in almost all other proposals for query evaluation under the well-founded semantics, e.g., the alternating fixpoint [10, 31] or various modifications of the SLD-resolution (global SLS-resolution [3, 14, 19] or SLG-resolution [4, 5]).

We define a closed-form bottom-up evaluation algorithm for Datalog$^{\mathcal{C},\neg}$ that computes the well-founded model using a constraint variation of the Alternating fixpoint method [31]:

Definition 2.7 (Alternating Fixpoint) *A* constraint atom *is a pair* (A, C) *where* $A(x_1, \ldots, x_k)$ *is a predicate symbol and* $C \in \mathcal{C}$ *such that* $FV(C) \subseteq FV(A)$*.*

A (constraint three-valued) interpretation *is a pair of sets of constraint atoms* (I^+, I^-) *such that if* $(A, C_1) \in I^+$ *and* $(A, C_2) \in I^-$ *then* $C_1 \wedge C_2$ *is not satisfiable.*

Let (I^+, I^-) *be an interpretation and* P *a* Datalog$^{\mathcal{C},\neg}$ *program. Then*

$$\begin{aligned} T_P^{\mathcal{C}}(I^+, I^-) = \{(Q, A) : \; & Q \leftarrow D, P_1, \ldots, P_k, \neg R_1, \ldots \neg R_l \in P, \\ & (P_i, B_i) \in I^+, (R_j, C_j) \in I^-, \\ & A_0 = D \wedge B_1 \wedge \ldots \wedge B_k \wedge C_1 \wedge \ldots \wedge C_l, \\ & A \in \exists_{FV(A)}(A_0), \\ & \textit{if } A \leq A' \textit{ for some } (Q, A') \in I^+ \textit{ then } A = A' \,\} \end{aligned}$$

is the (constraint based) immediate consequence operator *(where the variables in A, B_i, and C_j are renamed according to free variables of Q, P_i, and Q_j, respectively) and*

$$W_P^{\mathcal{C}}(I^+, I^-) = (\mathrm{lfp}\,\lambda x.T_P^{\mathcal{C}}(x, I^-), \neg\,\mathrm{lfp}\,\lambda x.T_P^{\mathcal{C}}(x, \neg I^+))$$

is the (constraint based) alternating fixpoint operator *(where $\neg$ is the complementation operation of $\mathcal{C}$ extended to sets constraint atoms). We define*

$$M_{\mathrm{WF}}^{\mathcal{C}}(P) = \mathrm{lfp}\,\lambda(x,y).W_P^{\mathcal{C}}(x,y).$$

Again, $W_P^{\mathcal{C}}$ is monotonic in both components and thus $M_{\mathrm{WF}}^{\mathcal{C}}(P)$ exists. Our definition of the alternating fixpoint separates the phase of computing the underestimates of true facts (left side) from the computing of overestimates of the not-false facts (right side). These two phases can be folded together into one alternating application of the $T_P^{\mathcal{C}}$ operator, similarly to [31]. To show faithfulness of $M_{\mathrm{WF}}^{\mathcal{C}}(P)$ w.r.t. $M_{\mathrm{WF}}(P)$ we compare the obtained results:

Definition 2.8 *Let I be a set of constraint atoms and J a three-valued constraint interpretation. We define*

$$\|I\| = \{A\theta : (A,C) \in I, \theta \models C\}, \qquad \|J\| = (\|J^+\|, \|J^-\|)$$

The $\|.\|$ map assigns the corresponding ground interpretation (the semantics) to every constraint interpretation. Using the properties of the constraint operations we can show:

Theorem 2.9 (Soundness and Completeness) *Let $\mathcal{C}$ be a constraint compact constraint class and P an arbitrary* Datalog$^{\mathcal{C},\neg}$ *program. Then $\|M_{\mathrm{WF}}^{\mathcal{C}}(P)\| = M_{\mathrm{WF}}(\mathrm{ground}(P))$.*
P r o o f: Immediate from the properties of the constraint operations and the relationship between lfp and gfp by induction on stages of $M_{\mathrm{WF}}(\mathrm{ground}(P))$ and $M_{\mathrm{WF}}^{\mathcal{C}}(P)$.

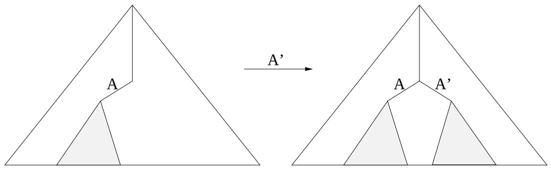
Note that the $M_{\mathrm{WF}}^{\mathcal{C}}(P)$ may not be unique due to the possibility of different constraint encodings of the same set of ground atoms. However, the semantics—the encoded set of ground atoms—is always unique. Moreover we can show

Theorem 2.10 (Termination) *Let $\mathcal{C}$ be a constraint compact constraint class. Then $M_{\mathrm{WF}}^{\mathcal{C}}(P)$ is finite for an arbitrary* Datalog$^{\mathcal{C},\neg}$ *program P.*
P r o o f: Follows from constraint compactness of $\mathcal{C}$.

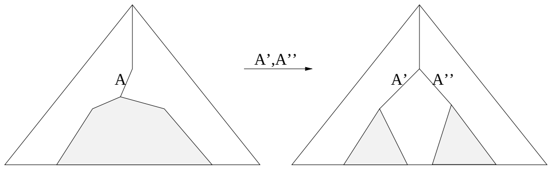
Thus we can compute $M_{\mathrm{WF}}^{\mathcal{C}}(P)$ bottom-up in finite time using nested loops.

3 Top-down evaluation of Datalog$^{\mathcal{C},\neg}$

The bottom-up algorithms are appealing by their simplicity and are often considered to be the basis for query evaluation under the well-founded semantics. However, there are several drawbacks connected with this approach (cf. Section 1), especially the algorithms unnecessarily recompute both the positive conclusions and the overestimates. Recomputing the positive conclusions can be



Extension by a new answer A'
not subsumed by any previous answer (A).



Refinement of A by
$A' \subset A$ and $A'' \subset A$ (simultaneously).

Note that while the pictures for both the transformations may seem similar, the effects are completely different: the first transformation creates new subtrees in an existing tree, while the second one takes an existing subtree and refines it. The second transformation may lead to complete elimination of subtrees in the refined SLD-resolution tree.

Fig. 1. Effect of extension and refinement on a SLD-resolution tree.

---

avoided by remembering the last positive iteration of the alternating fixpoint. However, the overestimates have to be recomputed every time—this is due to the anti-monotonicity of the associated operator and the inability to *remove* the no longer consistent overestimates.

In this section we introduce a new alternative method for evaluating the Datalog$^{\mathcal{C},\neg}$ programs. The technique is based on Constraint Memoing [27] and SLG-resolution [4] enhanced with a *refinement-based* handling of negative subgoals. The algorithm computes both the positive conclusions and overestimates concurrently using a constraint version of tabulated resolution [27]. The recomputation of the overestimates is avoided by creating *dependent* answers. These dependencies are later used by a refinement operator in order to keep the computation consistent. We present the method as a rewriting system over a set of nodes representing the current state of the derivation. The rules of the rewriting system perform two main tasks:

1. Extension of a *waiting* resolution node by a new answer.
2. Refinement of a *dependent* node by a previously derived answer.

While we do not necessarily represent the derivation in the form of a resolution tree, the two operations can be depicted by Figure 1. To define the rewriting system we need to introduce several additional notions. First, we need to represent the overestimates in such a way that they can be refined later:

Definition 3.1 (Conditional Constraint) *Let $C \in \mathcal{C}$ be a constraint and M a triple $[X, D, L]$ where X is a set of variables, $D \in \mathcal{C}$, and L a set of goals. Then $C[M]$ is a* conditional constraint.

We denote by $C[]$ a constraint where $M = [\{\}, true, \{\}]$. We also shorthand the maximal overestimate $\text{true}[FV(G), D, \{(G, D)\}]$ *created from a single subgoal (G, D) by $[G, D]$.*

We extend the constraint operations from Definition 2.1 to conditional constraints as follows:

Conjunction: *Let $C_1[X_1, D_1, L_1]$ and $C_2[X_2, D_2, L_2]$ be conditional constraints. The conjunction operation is defined by*

$$C_1[X_1, D_1, L_1] \wedge C_2[X_2, D_2, L_2] = C_1 \wedge C_2[X_1 \cup X_2, D_1 \wedge D_2, L_1 \cup L_2].$$

Projection: *Similarly, in the case of projection*

$$\exists_Y C[X, D, L] = \{C_i[X \cap Y, D \wedge C, L] : C_i \in \exists_Y(C)\}.$$

Complementation: *The role of the* complementation operator *is encapsulated in the* refinement operator *for dependent constraints. Let $C[X, D, L]$ be a conditional constraint such that $(G, E) \in L$ and $\mathbf{A}$ a set of constraints. Then we define*

$$\begin{aligned}\mathrm{rf}(C[X, D, L], (G, E), \mathbf{A}) = \{\ & C \wedge D_i[X, D \wedge D_i, L] : \\ & D_i \in \exists_X(D \wedge \textstyle\bigwedge_{A_j \in \mathbf{A}} \neg A_j), C \wedge D_i \textit{ is satisfiable }\ \}\end{aligned}$$

The refinement operator has no effect if $(G, E) \notin L$. We also omitted the renaming of the constraints in $\mathbf{A}$ with respect to the free variables of G.

The constraint subsumption does not have to be extended as it is never used for the conditional constraints (cf. Definition 3.3).

The conditional constraints are used to derive overestimates similarly to the right-hand side of the alternating fixpoint computation. However, the conditional part of the constraint allows us to *retract* inconsistent overestimates without the need for recomputation—it accumulates all the necessary information needed to perform such a refinement (cf. right part of Figure 1).

The rewriting rules of Constraint Memoing(WF) are defined over a finite set S of nodes that represents the current state of the derivation:

Definition 3.2 (Constraint Memoing(WF) Nodes) *Let G, H be atoms, C a constraint in $\mathcal{C}$, D a conditional constraint in $\mathcal{C}$, and $\mathbf{P}$ a set (list) of literals. Then we define the following nodes:*

$\mathsf{root}(G; C)$	new subtree to be resolved against the program P
$\mathsf{act}(G; C)$	goal $(G; C)$ is active
$\mathsf{body}(G; \mathbf{P}; D)$	current resolvent
$\mathsf{pos\_goal}(G; H, C; \mathbf{P}; D)$	positive literal H selected
$\mathsf{neg\_goal}(G; H, C; \mathbf{P}; D)$	negative literal $\neg H$ selected
$\mathsf{wait}(G; H, C; \mathbf{P}; D)$	waiting for answers to subquery (H, C)
$\mathsf{ans}(G; D)$	answer

With every *active goal* (G, C), i.e., $\mathsf{act}(G, C) \in S$, where S is the current state of the computation, we associate two sets of constraints:

Definition 3.3 (Answer Set) *Let* $\{\mathsf{ans}(G; A_i[L_i])\}$ *be all the answers derived from* $\mathsf{root}(G; C)$*. We define*

$$\begin{aligned}\text{ans}(G, C) = \{&A_i[] : A_i \not\leq A_j \text{ for any } \mathsf{ans}(G; A_j[]), 0 < j < i\} \cup \\ \{&A_i[L_i] : A_i \not\leq A_j \text{ for any } \mathsf{ans}(G; A_j[L_j]), 0 < j < i\}\end{aligned}$$

The set collects all the (so far) derived answers $\mathsf{ans}(G; A[])$ that are not subsumed by any previous answer (i.e., answer derived prior to the answer in consideration) and all the conditional answers $\mathsf{ans}(G; A[L])$ not subsumed by any definite answer or a previously derived conditional answer. To derive negative conclusions we have to construct sets of atoms that cannot be derived from a given Datalog$^{\mathcal{C},\neg}$ program consistently with the current state of the derivation—the constraint version of the $U_P(I)$ set. We characterize this set by a dual concept:

Definition 3.4 (Frontier of a Derivation) *We define*

$$\begin{aligned}\text{fr}(G, C) = \{D : &\text{ such that } \mathsf{ans}(G, D[L]) \text{ is derived from } \mathsf{root}(G; C) \\ &\text{ and is consistent with the current state of derivation}\}\end{aligned}$$

The intuition behind the use of the *frontier* (similarly to [23]) is to capture the frontier (non-failed) nodes in a representation of a SLD-resolution tree and assume they resolve to an empty clause without any additional restrictions (i.e., the most optimistic consistent overestimate). In this paper we use the simplest approach to deriving this set. However, more sophisticated methods [5] are immediately applicable. Now we can present the rewriting rules of Constraint Memoing(WF):

Definition 3.5 (Constraint Memoing(WF) Rewriting Rules) Let S be a finite set of Constraint Memoing(WF) nodes. The rewriting system contains following rules:

Clause Resolution (CR). First we define a rule that resolves a new goal (H, C) against the clauses of the given Datalog$^{\mathcal{C},\neg}$ program P:

$$\frac{\mathsf{root}(H; C)}{\mathsf{body}(H; \mathbf{P_i}; C \wedge D_i[])} \qquad \begin{array}{l}\text{for all } (H \leftarrow D_i, \mathbf{P_i}) \in P \\ \text{where } C \wedge D_i \text{ is satisfiable}\end{array}$$

Selection (SL). The rewriting system can use an arbitrary selection rule; the following two rules represent selection of a positive literal or a negative literal:

$$\frac{\mathsf{body}(H; \mathbf{P}; C[L])}{\mathsf{pos\_goal}(H; P, C_i; \mathbf{P} - \{P\}; C[L])} \qquad \begin{array}{l}\text{for } P \in \mathbf{P} \text{ selected,} \\ \text{and } C_i \in \exists_{FV(P)} C\end{array}$$

$$\frac{\mathsf{body}(H; \mathbf{P}; C[L])}{\mathsf{neg\_goal}(H; R, C_i; \mathbf{P} - \{\neg R\}; C[L])} \qquad \begin{array}{l}\text{for } \neg R \in \mathbf{P} \text{ selected,} \\ \text{and } C_i \in \exists_{FV(R)} C\end{array}$$

Answer Projection (AP). In the case the resolvent is empty (we have resolved all the subgoals of the original clause, if any) we determine the portions of the constraint C relevant to the head of the clause H and project the resulting constraint to the free variables of the head.

$$\frac{\mathsf{body}(H; \{\}; C[L])}{\mathsf{ans}(H; C_i[L_i])} \qquad \text{for } C_i[L_i] \in \exists_{FV(H)} C[L]$$

Positive Goal (PG). The rules for positive goal processing test if a goal at least as general as the selected goal has already been encountered. In that case we reuse the answers obtained for the more general goal:

$$\frac{\mathsf{pos\_goal}(H; P, D; \mathbf{P}; C[L])}{\mathsf{wait}(H; P, D'; \mathbf{P}; C[L])} \qquad \begin{array}{l}\text{if } \mathsf{act}(P, D') \text{ such that}\\ D \leq D' \text{ does exists}\end{array}$$

Otherwise we initiate a new computation for the selected goal:

$$\frac{\mathsf{pos\_goal}(H; P, D; \mathbf{P}; C[L])}{\mathsf{wait}(H; P, D; \mathbf{P}; C[L]), \mathsf{root}(P; D), \mathsf{act}(P, D)} \qquad \begin{array}{l}\text{if } \mathsf{act}(P, D') \text{ such that}\\ D \leq D' \text{ does not exist}\end{array}$$

Answers Resolution (AR). This rule is responsible for propagating new answers to resolvents waiting for these answers:

$$\frac{\mathsf{wait}(H; P, D; \mathbf{P}; C[L])}{\mathsf{wait}(H; P, D; \mathbf{P}; C[L]), \mathsf{body}(H; \mathbf{P}; C[L] \wedge A[M])} \qquad \begin{array}{l}\text{for } A[M] \in \text{ans}(P, D)\\ \text{where } C \wedge A \text{ is satisfiable}\end{array}$$

Note that every answer in the $\text{ans}(P, D)$ set is propagated exactly *once* by the AR rule for every wait node.

Negative Goal (NG). The rules for processing a negative subgoal are similar to the rules for the positive goal. The main difference is, that instead of waiting for answers to the negative subgoal, we *weaken* the conclusion by creating an dependent node. This is due to the difference between handling of the positive and negative dependencies[7]:

$$\frac{\mathsf{neg\_goal}(H; P, D; \mathbf{P}; C[L])}{\mathsf{body}(H; \mathbf{P}; C[L] \wedge [P, D])} \qquad \begin{array}{l}\text{if } \mathsf{act}(P, D') \text{ such that}\\ D \leq D' \text{ does exists}\end{array}$$

$$\frac{\mathsf{neg\_goal}(H; P, D; \mathbf{P}; C[L])}{\mathsf{body}(H; \mathbf{P}; C[L] \wedge [P, D]), \mathsf{root}(P; D), \mathsf{act}(P, D)} \qquad \begin{array}{l}\text{if } \mathsf{act}(P, D') \text{ such that}\\ D \leq D' \text{ does not exist}\end{array}$$

We call the rules introduced so far (CR, SL, AP, PG, AR, NG) the *extension* rules.

In the processing of negative subgoals we have introduced dependent nodes—the overestimates. To maintain consistency we need to use these dependencies when new answers are derived. There are two cases:

[7] This phenomenon can be summarized by "$p \leftarrow p$ implies p is false" while "$p \leftarrow \neg p$ implies p is undefined".

Refinement (RF). If a definite answer (i.e., an independent answer) is obtained, we need to refine all nodes where we assumed the opposite. We present the rules that handle this situation using the following metarule

$$\frac{\mathsf{node}(C[L])}{\mathsf{node}(C_i[L_i])} \qquad \begin{array}{l} C_i[L_i] \in \mathrm{rf}(C[L], (G, C), \{A\}) \\ \text{and } A[] \in \mathrm{ans}(G, C) \end{array}$$

where $\mathsf{node}(C[L])$ is an arbitrary node containing the dependent constraint $C[L]$. This rule maintains the consistency of the definite answers and the overestimates.

Dependency Elimination (DE). The other case is when we determine definite falsity of some atoms. Then we completely eliminate the dependency using this information:

$$\frac{\mathsf{node}(C[L])}{\mathsf{node}(C[L]), \mathsf{node}(C_i[X_i, D_i, L_i - \{(G, C)\}])} \qquad \begin{array}{l} \text{where } C_i[X_i, D_i, L_i] \in \\ \quad \mathrm{rf}(C[L], (G, C), \mathrm{fr}(G, C)) \end{array}$$

In the case $C \leq^{\mathcal{C}} C_i$ for some i we can completely eliminate the original node[8]. The RF rules correspond to the complementation of the definite answers in the alternating fixpoint computation, and the DE rules to the complementation of the overestimates.

A *derivation* for the query (G, C) is built from the *initial system*

$$S = \{\mathsf{act}(G; C), \mathsf{root}(G; C)\}$$

by repeated application of the above rules as long as S changes. We call S a *final system* if no rule that changes S can be applied.

The only remaining problem is to determine the $\mathrm{fr}(G, C)$ sets. We use the following lemma:

Lemma 3.6 *Let S be a system in which no extension rule can be applied. Then for any goal G, C $\mathrm{fr}(G, C) = \{A : A[L] \in \mathrm{ans}(G, C)\}$.*

It is easy to see that the rewriting system terminates for any $\mathrm{Datalog}^{\mathcal{C},\neg}$ program:

Theorem 3.7 (Termination) *Let $\mathcal{C}$ be a constraint compact constraint class and P an arbitrary* $\mathrm{Datalog}^{\mathcal{C},\neg}$ *program. Then all Constraint Memoing(WF) rewriting sequences are finite.*

P r o o f: The sets $\mathrm{ans}(G, C)$ are finite for any (G, C) and every constraint can be non-trivially refined only finitely many times.

The rewriting system is not confluent. However, we can show that constraints in the computed nodes always *encode* the same set of ground atoms. We call such system *confluent up to* $\|.\|$.

Lemma 3.8 *The Constraint Memoing(WF) rewriting is confluent up to* $\|.\|$.

P r o o f: (sketch) Most of the cases are a consequence of results for standard SLD resolution. We only need to show that the refinement (RF) and dependency

[8] This is always the case for stratified $\mathrm{Datalog}^{\mathcal{C},\neg}$ programs.

elimination (DE) rules commute with the extension rules. The most complex case is for the answer resolution rule (AR):

$$\begin{tikzcd}
\{\mathsf{wait}(G; P, D; \mathbf{P}; C), \mathsf{ans}(P, A[L])\} \arrow[r, "(RF)"] \arrow[d, "(AR)"] & \{\mathsf{wait}(G; P, D; \mathbf{P}; C), \mathsf{ans}(P, A_i[L_i])\} \arrow[d, "(AR)"] \\
\{\mathsf{body}(G; \mathbf{P}; C \wedge A[L])\} \arrow[r, "(RF)"] & \{\mathsf{body}(G; \mathbf{P}; C \wedge A_i[L_i])\}
\end{tikzcd}$$

Commutativity of the diagram follows from commutativity and associativity of conjunction. The remaining cases are similar.

Now we can show the soundness and completeness of the proposed method:

Theorem 3.9 (Soundness and Completeness) *Let $\mathcal{C}$ be a constraint compact constraint class, P a* Datalog$^{\mathcal{C},\neg}$ *program, S a final system for a query (Q, C), and θ a ground substitution such that $\theta \models C$. Then*

$$\begin{array}{l}
Q\theta \in I^+ \iff \theta \models D \textit{ for some } D[] \in \mathrm{ans}(Q, C) \\
Q\theta \in I^- \iff \theta \not\models D \textit{ for all } D[L] \in \mathrm{ans}(Q, C)
\end{array}$$

where $(I^+, I^-) = M_{\mathrm{WF}}(\mathrm{ground}(P))$ *and* $\mathrm{ans}(Q, C) \subset S$.
P r o o f: It is sufficient that the equivalence holds for an arbitrary reduction sequence. This can be done by induction on stages of the $W_{\mathrm{ground}(P)}$ (only if part) and by the relative "age" of the ans nodes (if part), similarly to [27]. The result then follows from Theorem 3.7 and Lemma 3.8.

By inspecting the rules we can also see that (for an uninstantiated query) the extension rules simulate the two phases of the alternating fixpoint (concurrently) while the RF and DE rules maintain consistency of the definite answers and the overestimates. Thus:

Theorem 3.10 (Complexity) *Let $\mathcal{C}$ be a constraint compact constraint class and $f(P)$ the number of steps of the alternating fixpoint algorithm*[9]*. Then the number of rewriting steps is at most $3f(P) + c$.*

This bound holds even if the original program P is rewritten using the MST [13], assuming the SIPS [2] used in the MST transformation matches the selection rule (SL).

4 Implementation Issues Future Work

In this section we consider several implementation issues connected with the proposed evaluation method.

[9] It is the sum of the number of operations in the internal loops over all the iterations of the external loop.

Refinement. The refinement rules (RF) are defined for all dependent nodes. However, there are several modifications to this general strategy that preserve correctness:

- It is sufficient to use the RF rules only for the wait, body, and ans nodes. This modification does not affect any of the properties of the query evaluation as all other nodes can be rewritten to one of these in one step without any changes to the dependent constraint.
- It is even sufficient to use the RF rules for the ans nodes only (a lazy approach)[10]. While this approach preserves correctness, it affects the level of constraint propagation during the query evaluation: while the number of refined nodes drops, it may lead to unnecessary computation of inconsistent answers using the extension rules (due to lazy constraint propagation).

Dependent Constraint Management. Note that the refinement affects the set $\mathrm{ans}(G, C)$: an answer that has been subsumed before an application of the RF rule may not be subsumed after the rule finishes. This leads to the need of chaining the subsumed answers and re-checking for subsumption. However, the subsumption is a crucial and necessary feature that guarantees termination—expanding all overestimates (similarly to SLG-resolution[11]) leads immediately to non-terminating rewritings.

Other problem with the dependent constraints is that the dependency may grow in size with the growing distance from the point where it was introduced by the NG rule—essentially we have to *remember* how a given constraint has been derived using the overestimate. This situation can be avoided by splitting the dependency at the points of answer resolution (AR). The resulting method is equivalent to the proposed one and ensures bounded size of the overestimates.

Frontier Management. Lemma 3.6 shows that the set $\mathrm{fr}(G, C)$ can be determined when all possible overestimates have been computed. This is analogous to computing the complement of all the overestimates in the alternating fixpoint approach. However, an *incremental* version of the DE rule would be preferable and is in the focus of future research. Also, using the *goal completion detection* technique [5] we can determine the frontier sets for completed goals on fly. This is especially important in the case of stratified Datalog$^{\mathcal{C},\neg}$ programs. Under this assumption we can show:

Theorem 4.1 (Stratified Datalog$^{\mathcal{C},\neg}$) *Let P be a modularly stratified Datalog program [20]. Then the rewriting system (under a depth-first rewriting strategy) will not use the RF rules in the derivation of answers except for the* body *nodes immediately succeeding the* neg_goal *nodes and the DE rules become trivial: in the lower stratum no dependent nodes remain.*

[10] Follows from commutativity and associativity of conjunction.

[11] SLG resolution expands all the dependent answers. However, termination is preserved by requiring the negative subgoals to be ground and testing for variants of the dependencies.

This fact follows from the observation that the whole lower stratum is completed before the *caller* node needs to be expanded by an overestimate. Thus the evaluation is complexity-wise no worse than the bottom-up evaluation of stratified Datalog. Note that this is not true for the lazy approaches, e.g., the SLG resolution [4].

Safe Datalog, Floundering Queries. The other problem we face is the requirement of a total complementation operation on the constraint class. This is often not possible or feasible. Thus we look for restricted versions of complementation, e.g., the set difference in relational algebra. Clearly a *difference operator* can be used in the refinement operator instead of complementation. However, the difference may not be defined for all pairs of constraints, (if it were, we would be able to define complementation using the true constraint). There are two solutions:

the "database" solution: we syntactically restrict the class of queries to *allowed queries*—those for which the difference is guaranteed to be total. This approach is usually taken by the deductive systems [1, 18].

the "logic programming" solution: we check for definedness of the difference at run-time and signal an error if the difference is undefined. This approach is taken by most of the resolution-based methods—they term this situation *query floundering* [3, 4, 5, 19].

Example 4.2 In [18] a restricted (safe) Datalog with integer gap-order constraints was introduced. The safety criterion essentially states that only unary gap-order constraints (i.e., constraints of the form $x < c$ or $x > c$) may *pass through negation*—appear as answers to literals used under negation in the query. [17] shows that the class of gap-order constraints is constraint compact and the safety restriction of [18] ensures closure under the limited complementation. Thus we can use our technique to evaluate safe Datalog queries over gap-order constraints.

Selection Rules, Search Strategies. The proposed method is very flexible: it does not impose any particular order on the application of the rewriting rules while maintaining correctness. This leads an open door to optimization of the query evaluation. Note that program transformation based techniques (e.g., [13, 16, 22]) transform the original program *before* the evaluation starts. Thus the SIPS is fixed during the evaluation. Our method allows the use of adaptive search strategies and selection rules to improve the evaluation efficiency.

Extension to Well-founded Semantics for Aggregation. The proposed evaluation technique can be also extended to evaluate the Well-founded Semantics of Datalog queries with aggregation [24]. The delaying technique used for the negative literals can be used for the literals under an aggregation operator. Similarly, the refinement operation remains unchanged. The main problem of incorporating aggregation lies in the determining of the *frontier* in the presence of aggregation. However, the bottom-up methods face the same problem (cf. [24] for discussion of Well-founded Semantics for Aggregation).

5 Conclusion

We introduced a new query evaluation method based on tabulated resolution for computing the well-founded semantics for Datalog$^{\mathcal{C},\neg}$ programs and shown its correctness and termination.

Related work. There are two other proposals for top-down evaluation of logic programs with negation close to our work: First, the SLG resolution [4] designed to compute the well-founded model for logic programs. However, SLG does not allow non-ground negative subgoals and this restriction can not be easily lifted by introducing of constraints closed under complements: SLG does not terminate if non-ground negative literals are introduced. The other approach [23] is also in the area of general logic programs and thus termination can not be guaranteed. Moreover, this approach is based on resolution without tabulation which leads to unnecessary recomputation of known derivations. However, it introduces an interesting way of computing the $\mathrm{fr}(G, C)$ sets.

On the other hand there are several proposals for computing the well-founded model bottom-up. They are usually based on the alternating fixpoint technique, e.g., [15]. The only exception is the Well-founded Ordered Search technique [22]. Note that while the Well-founded Ordered Search restricts the use of the alternating fixpoint to minimum, it can not avoid it completely. Our technique avoids the alternating fixpoint computation altogether by introducing the overestimates and using the refinement rules (see [22] for comparison of SLG resolution with Well-founded Ordered Search). Moreover, all the deductive systems that support well-founded semantics are restricted to allowed queries and thus the techniques are not directly applicable to the constraint case.

References

1. Abiteboul, S., Hull, R., Vianu, V. Foundations of Databases. Addison–Wesley, 1995.
2. Beeri, C., Ramakrishnan, R. On the Power of Magic. Proc. *ACM–PODS* 1987, 21–37.
3. Bol, R., Degerstedt, L. Tabulated Resolution for Well Founded Semantics. Proc. *ILPS* 1993.
4. Chen, W., Warren, D. S. Query Evaluation under the Well Founded Semantics. Proc. *ACM–PODS* 1993: 168-179
5. Chen, W., Swift, T., Warren, D. S. Efficient Top-Down Computation of Queries under the Well-Founded Semantics. JLP 24(3): 161-199 (1995).
6. Freire J., Swift, T., Warren, D. S. Taking I/O Seriously: Resolution Reconsidered for Disk. Manuscript. SUNY at Stony Brook, 1996.
7. Jaffar J., Maher, M. J. Constraint Logic Programming: A Survey. J. Logic Programming 1994, 19. 20:503–581.
8. Kanellakis, P. C., Goldin, D. Constraint Programming and Database Query Languages. Proc. *2nd TACS*, 1994.
9. Kanellakis, P. C., Kuper, G. M., Revesz, P.Z . Constraint Query Languages. Journal of Computer and System Sciences 51(1):26-52, 1995.

10. Kemp, D. B., Srivastava, D., Stuckey, P. J. Bottom-up evaluation and query optimization of well-founded models. *Theoretical Computer Science* 146 (1995) 145–184.
11. Sagonas, K., Swift, T., Warren, D. S. XDB as an Efficient Deductive Database Engine. Proc. *1994 ACM SIGMOD Intl. Conf. on Management of Data*, pp. 442–453, 1994.
12. Langford, C. Some Theorems on Deducibility. *Annals of Mathematics*, vol. 28, 16–40, 459–471, 1927.
13. Morishita, S. An alternating fixpoint tailored to magic programs. Proc. *ACM–PODS* 1993: 123–134.
14. Przymusinski, T. C. Every Logic Program Has a Natural Stratification And an Iterated Least Fixed Point Model. Proc. *ACM–PODS* 1989: 11-21
15. Ramakrishnan, R., Srivastava, D., Sudarshan, S. CORAL: Control, relations, and logic. Proc. *18th VLDB*, 238–249, 1992.
16. Ramakrishnan, R., Srivastava, D., Sudarshan, S. Controlling the Search in Bottom-up evaluation Proc. *JICSLP'92*, 273–287, 1992.
17. Revesz, P. Z. A Closed Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science*, vol. 116, no. 1, 117–149, 1993.
18. Revesz, P. Safe Stratified Datalog with Integer Order Programs. In *Proc. First International Conference on Constraint Programming*, Montanari U., Rossi F. eds., Springer-Verlag LNCS 976, Cassis, France, September 1995.
19. Ross, K. A. A Procedural Semantics for Well Founded Negation in Logic Programs. Proc. *ACM–PODS* 1989: 22-33.
20. Ross, K. A. Modular Stratification and Magic Sets for Datalog Programs with Negation. Proc *ACM–PODS* 1990, 161–171.
21. Srivastava, D., Ramakrishnan, R., Revesz, P. Z. Constraint Objects. Proc. *Intl. Workshop on Principles and Practice of Constraint Programming*, 218–228, 1994.
22. Stuckey, P., Sudarshan, S. Well-Founded Ordered Search. *Foundations of Software Technology and Theoretical Computer Science*, 1993.
23. Stuckey, P. Negation in Constraint Logic Programming. *Information and Computation*, 118(1): 12–33, 1995.
24. Sudarshan S., Srivastava, D., Ramakrishnan R., Beeri, C. Extending the Well-Founded and Valid Semantics for Aggregation. Proc. *ILPS'93*, 590–608, 1993.
25. Swift, T., Warren, D. Analysis of SLG-WAM Evaluation of Definite Programs. Proc. *1994 International Logic Programming Symposium*, 1994, 219–235.
26. Toman, D., Chomicki, J., Rogers D. S. Datalog with Integer Periodicity Constraints. Proc. *1994 International Logic Programming Symposium*, 1994, 189–203.
27. Toman, D. Top-Down beats Bottom-Up for Constraint Based Extensions of Datalog. Proc. *1995 International Logic Programming Symposium*, MIT Press, 1995, 98-112.
28. Ullman J. D. Principles of Database and Knowledge-base Systems, Vol. 1,2. Computer Science Systems, 1989.
29. Van Gelder, A., Ross, K. A., Schlipf, J. S. Unfounded Sets and well-founded semantics for general logic programs. Proc. *ACM Symposium on Principles of Database Systems*, 1988.
30. Van Gelder, A., Ross, K. A., Schlipf, J. S. The Well-Founded Semantics for General Logic Programs. JACM 38(3): 620-650 (1991).
31. van Gelder, A. The Alternating Fixpoint of Logic Programs with Negation. Proc. *ACM–PODS* 1989: 1-10.