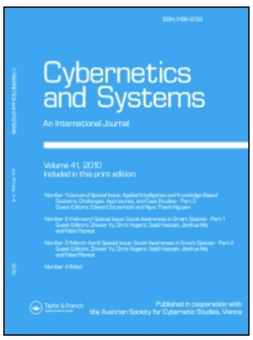

Taylor & Francis
Taylor & Francis Group

Cybernetics and Systems
An International Journal

ISSN: 0196-9722 (Print) 1087-6553 (Online) Journal homepage: http://www.tandfonline.com/loi/ucbs20

Extending Query-Subquery Nets for Deductive Databases under the Well-Founded Semantics

Son Thanh Cao, Linh Anh Nguyen & Ngoc Thanh Nguyen

To cite this article: Son Thanh Cao, Linh Anh Nguyen & Ngoc Thanh Nguyen (2017) Extending Query-Subquery Nets for Deductive Databases under the Well-Founded Semantics, Cybernetics and Systems, 48:3, 249-266

To link to this article: http://dx.doi.org/10.1080/01969722.2016.1276777

CYBERNETICS AND SYSTEMS: AN INTERNATIONAL JOURNAL
2017, VOL. 48, NO. 3, 249–266
http://dx.doi.org/10.1080/01969722.2016.1276777

Taylor & Francis
Taylor & Francis Group

Extending Query-Subquery Nets for Deductive Databases under the Well-Founded Semantics

Son Thanh Cao[a], Linh Anh Nguyen[b,c], and Ngoc Thanh Nguyen[d]

[a]Faculty of Information Technology, Vinh University, Vinh, Nghe An, Vietnam; [b]Division of Knowledge and System Engineering for ICT, Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam; [c]Institute of Informatics, University of Warsaw, Warsaw, Poland; [d]Department of Information Systems, Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wroclaw, Poland

ABSTRACT

We propose a method, called QSQN-WF, for evaluating queries to Datalog$^\neg$ databases under the well-founded semantics. It is the first one that is set-at-a-time and strictly goal-directed w.r.t. SLS-resolution defined by Przymusinski. These properties are important for reducing accesses to the secondary storage and redundant computations. The first property distinguishes our method from the one based on SLG-resolution by Chen, Swift, and Warren (1995) (which is tuple-at-a-time). The second property distinguishes our method from the ones based on the magic-sets transformation by Kemp, Srivastava, and Stuckey (1995) and Morishita (1996), which use magic atoms not in the most appropriate way and are not strictly goal-directed w.r.t. SLS-resolution. Our method follows SLS-resolution, with Van Gelder's alternating fixpoint semantics on the background, but uses a query-subquery net to implement tabulation and the set-at-a-time technique, reduce redundant computations, and allow any control strategy within each iteration of the main loop. It is sound and complete w.r.t. the well-founded semantics and has PTIME data complexity.

KEYWORDS

Datalog with negation; deductive databases; query processing; query-subquery nets; well-founded semantics

Introduction

Datalog$^\neg$ is an important query language, as it expresses the class of all PTIME queries on ordered databases (see, e.g., Abiteboul, Hull, and Vianu 1995) when used either with the noninflationary semantics or the well-founded semantics. The well-founded semantics is a better choice for Datalog$^\neg$, as it always gives a unique intended model for a Datalog$^\neg$ database and coincides with the standard semantics for stratified deductive databases.

Datalog$^\neg$ programs are normal logic programs without function symbols and are usually assumed to be safe in the sense that every variable occurring in the head or a negative literal in the body of a program clause also occurs in

CONTACT Linh Anh Nguyen nguyenanhlinh@tdt.edu.vn Division of Knowledge and System Engineering for ICT, Faculty of Information Technology, Ton Duc Thang University, No. 19 Nguyen Huu Tho Street, Tan Phong Ward, District 7, Ho Chi Minh City, Vietnam.

a positive literal in its body. The well-founded semantics was originally introduced in (Gelder, Ross, and Schlipf 1991) for normal logic programs. In (Gelder 1993), the author characterized the well-founded semantics by the alternating fixpoint. As calculi for normal logic programs that are sound and complete w.r.t. the well-founded semantics, there are SLS-resolution (Przymusinski 1989), global SLS-resolution (Ross 1992), SLG-resolution (Chen, Swift, and Warren 1995), and SLDFA-resolution (Drabent 1995). Among these calculi, SLS-resolution is most closely related to the alternating fixpoint characterization.

For the implementation issue of query evaluation for normal logic programs under the well-founded semantics, Chen, Swift, and Warren (1995) presented efficient techniques for implementing SLG-resolution, which maintain positive and negative dependencies among subgoals in a top-down evaluation. The method proposed by Chen, Swift, and Warren (1995), however, is tuple-at-a-time but not set-at-a-time and thus not suitable for evaluating queries to Datalog$^\neg$ databases. It relies on detecting positive and negative loops, delaying subgoals when possible loops occur, checking completion of subgoals and resuming their activeness when possible. It follows depth-first computation and maintains a stack of subgoals as in Prolog. All of these techniques are tuple-oriented and it is hard to convert the method to a set-oriented one.

Some authors (Kemp, Srivastava, and Stuckey 1995; Morishita 1996) proposed bottom-up evaluation methods for Datalog$^\neg$ under the well-founded semantics. Their methods are based on the magic-sets transformation and Van Gelder's alternating fixpoint characterization. As discussed below, they have certain drawbacks.

The magic-sets transformation (Beeri and Ramakrishnan 1991) is a tabulation technique originally formulated for Datalog (without negation). It simulates the top-down QSQR (query-subquery recursive) evaluation (Vieille 1989) by rewriting the considered program together with the given query to another equivalent one that can then be evaluated using a bottom-up technique (e.g., the improved semi-naive evaluation). The magic-sets transformation is usually formulated using adornments. It simulates SLD-resolution (defined for positive logic programs) in pushing constants from goals to subgoals. A strict simulation of SLD-resolution is obtained only when the magic-sets technique is used together with annotations for pushing variable repeats from goals to subgoals (see, e.g., Abiteboul, Hull, and Vianu 1995). Also note that magic relations consist of tuples of constants and are less general than "input" relations, which consist of tuples of terms possibly with variables. Thus, the corresponding subsumption is not checked, which may lead to redundant computations and waste of memory.

The evaluation methods proposed for Datalog$^\neg$ under the well-founded semantics (Kemp, Srivastava, and Stuckey 1995; Morishita 1996) use the magic-sets transformation with adornments but without annotations. The

method by Kemp, Srivastava, and Stuckey (1995) applies the alternating fixpoint computation for the transformed program in a differential way for truth facts. The method by Morishita (1996) uses a modified version of the alternating fixpoint computation, where the sequence of underestimates does not always increase. Both of the methods have the drawback that they use magic atoms not in the most appropriate way. Namely, magic atoms for deriving failures are used also for deriving successes, and conversely, magic atoms originated from the seed for deriving successes are used also for deriving failures. Thus, the mentioned methods are not strictly goal-directed w.r.t. SLS-resolution.

Query processing for Datalog$^{\neg}$ under the well-founded semantics is an important topic due to practical applications. As discussed above, the previously known evaluation methods for that have certain drawbacks and it is worth doing further research on the topic.

In this article, we extend query-subquery nets to formulate a method, called QSQN-WF, for evaluating queries to Datalog$^{\neg}$ databases under the well-founded semantics. Query-subquery nets were introduced by Nguyen and Cao (2012). Using such nets they proposed a method, called QSQN, for evaluating queries to Horn knowledge bases. The method is goal-directed, set-at-a-time, and has been designed so that the query processing is divided into appropriate steps which can be delayed to maximize adjustability, allow various control strategies, and reduce the redundant recomputation as much as possible. Cao and Nguyen (2015) developed the QSQN-TRE evaluation method as an extension of QSQN with tail-recursion elimination. Cao (2015) also extended QSQN to obtain the QSQN-STR method for evaluating queries to stratified knowledge bases. We refer the reader to (Cao 2016a) for further details on the mentioned methods.

Our method QSQN-WF provided in this article is the first evaluation method for Datalog$^{\neg}$ databases under the well-founded semantics that is set-at-a-time and strictly goal-directed w.r.t. SLS-resolution. These properties are important for reducing accesses to the secondary storage and redundant computations. Our method follows SLS-resolution, with Van Gelder's alternating fixpoint semantics on the background, but uses a query-subquery net to implement tabulation and the set-at-a-time technique, reduce redundant computations, and allow any control strategy within each iteration of the main loop. It is sound and complete w.r.t. the well-founded semantics and has PTIME data complexity.

The rest of this article is structured as follows. The next section recalls the most important concepts and definitions that are related to our work. After that, we present our QSQN-WF evaluation method. We then provide proofs of soundness, completeness, and PTIME data complexity of the method. Conclusions and a plan for future work are presented at the end. In addition, the related functions and procedures used for QSQN-WF are presented in the appendix.

Preliminaries

We assume that the reader is familiar with the basic notions of first-order logic such as *substitution, unification, positive logic program, goal, Horn knowledge bases, stratified negation,* and related ones. In this section, we recall only the most important definitions and notions that are needed for our work and refer the reader to other works (Lloyd 1987; Abiteboul, Hull, and Vianu 1995; Madalinska-Bugaj and Nguyen 2012) for further reading.

A *term* is either a constant or a variable. An *atom* is an expression of the form $p(t_1, \ldots, t_n)$, where $n \geq 0$, p is an n-ary predicate and each t_i is a term. If an atom contains no variables, it is called a *ground* atom. A *literal* is either an atom (called a *positive literal*) or the negation of an atom (called a *negative literal*). Each predicate is classified either as *intensional* or as *extensional*. An *expression* is either a term, a tuple of terms, or a formula (or a list of formulas) without quantifiers. A *simple expression* is either a term or an atom.

A *substitution* is a finite set $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$, where $x_1, \ldots, x_k$ are pairwise distinct variables, $t_1, \ldots, t_k$ are terms, and $t_i \neq x_i$ for all $1 \leq i \leq k$. The set $dom(\theta) = \{x_1, \ldots, x_k\}$ is called the *domain* of θ. The set $range(\theta) = \{t_1, \ldots, t_k\}$ is called the *range* of θ. The empty substitution is denoted by ε. The restriction of a substitution θ to a set X of variables is the substitution $\theta_{|X} = \{(x/t) \in \theta \mid x \in X\}$.

Let E be an expression and $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$ be a substitution. The *instance* of E by θ, denoted by $E\theta$, is the expression obtained from E by simultaneously replacing all occurrences of the variable x_i in E by the term t_i, for all $1 \leq i \leq k$.

Let $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$ and $\delta = \{y_1/s_1, \ldots, y_h/s_h\}$ be substitutions (where $x_1, \ldots, x_k$ are pairwise distinct variables, and $y_1, \ldots, y_h$ are also pairwise distinct variables). The *composition* of θ and δ, denoted by $\theta\delta$, is the substitution obtained from the sequence $\{x_1/(t_1\delta), \ldots, x_k/(t_k\delta), y_1/s_1, \ldots, y_h/s_h\}$ by deleting any binding $x_i/(t_i\delta)$ for which $x_i = (t_i\delta)$ and deleting any binding y_j/s_j for which $y_j \in \{x_1, \ldots, x_k\}$.

A substitution $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$ is *idempotent* if none of $x_1, \ldots, x_k$ occurs in any $t_1, \ldots, t_k$, which means $\theta\theta = \theta$. If θ and δ are substitutions such that $\theta\delta = \delta\theta = \varepsilon$, then we call them *renaming substitutions*. An expression E is a *variant* of an expression E' if there exist substitutions θ and γ such that $E = E'\theta$ and $E' = E\gamma$.

A substitution θ is *more general* than a substitution δ if $\delta = \theta\gamma$ for some substitution γ. Let Γ be a set of simple expressions. A substitution θ is called a *unifier* for Γ if $\Gamma\theta$ is a singleton. A unifier θ for Γ is called a *most general unifier* (mgu) for Γ if θ is more general than every unifier of Γ.

We denote the set of variables occurring in E by $Vars(E)$, where E is an expression or a substitution. A *fresh variant* of a formula φ, where φ can be an atom, a goal, or a program clause, is a formula $\varphi\theta$, where θ is a renaming

substitution such that $dom(\theta) = Vars(\varphi)$ and $range(\theta)$ consists of variables that were not used in the computation.

Datalog¬

In this subsection, we recall the Datalog¬ language and related notions.

Definition 1 (Safe Datalog¬ Program)

A *safe Datalog¬ program clause* (*w.r.t. the leftmost selection function*) is an expression of the form $A \leftarrow B_1, \ldots, B_k$ with $k \geq 0$, such that:

- A is an atom and each B_i is a literal, where negation is denoted by $\sim$ instead of $\neg$,
- every variable occurring in A also occurs in $(B_1, \ldots, B_k)$,
- every variable occurring in a negative literal B_j also occurs in some positive literal B_i with $i < j$.

The atom A is called the head and $(B_1, \ldots, B_k)$ the body of the program clause. A *safe Datalog¬ program* (*w.r.t. the leftmost selection function*) is a finite set of safe Datalog¬ program clauses.

A safe Datalog¬ program without negative literals in the clauses' bodies is called a *safe Datalog program*. From now on, by a Datalog¬ (resp. Datalog) program we mean a safe Datalog¬ (resp. Datalog) program.

Definition 2 (Extensional Instance)

An *instance* of extensional predicates is a mapping I that associates each extensional n-ary predicate p to a finite set $I(p)$ of n-ary tuples of constants. Sometimes, I is treated as the set $\{p(\bar{t}) \mid \bar{t} \in I(p)\}$. The size of I is defined to be the cardinality of this set.

Definition 3 (Datalog¬ Database)

A *Datalog¬ database* is defined to be a pair (P, I), where P is a Datalog¬ program for defining intensional predicates and I is an instance of extensional predicates.

In what follows, let (P, I) be a Datalog¬ database.

Definition 4 (Herbrand Base and Herbrand Interpretation)

- The *Herbrand universe* of (P, I), denoted by $U_{P,I}$, is the set of all constants occurring in (P, I).
- The *Herbrand base* of (P, I), denoted by $B_{P,I}$, is the set of all ground atoms of the form $p(t_1, \ldots, t_n)$, where p is a predicate used in (P, I) and each t_i belongs to $U_{P,I}$.
- A *Herbrand interpretation* for (P, I) is a subset of $B_{P,I}$.

Let M be a Herbrand interpretation. If $p(\bar{t})$ is a ground atom, then

$$M(p(\bar{t})) \stackrel{def}{\equiv} p(\bar{t}) \in M,$$

$$M(\sim p(\bar{t})) \stackrel{def}{\equiv} p(\bar{t}) \notin M.$$

Definition 5 (Immediate Consequence Operator)
Let *ground* $(P \cup I)$ be the set of all ground instances of clauses in $P \cup I$ and M a Herbrand interpretation for (P, I). The *immediate consequence operator* of (P, I), denoted by $T_{P,I}$, is defined on M as follows:

$$T_{P,I}(M) = \{A \mid A \leftarrow B_1, \ldots, B_k \in \text{ ground } (P \cup I) \text{ and } M(B_i) \text{ holds for all } 1 \leq i \leq k\}.$$

Let $T_{P,I} \uparrow \omega$ be defined as follows:

$$T_{P,I} \uparrow 0 = I,$$

$$T_{P,I} \uparrow (n+1) = T_{P,I}(T_{P,I} \uparrow n) \cup T_{P,I} \uparrow n, \text{ for } n \in \mathbb{N}$$

$$T_{P,I} \uparrow \omega = \bigcup_{n=0}^{\omega} T_{P,I} \uparrow n.$$

The Well-Founded Semantics of Datalog¬

Let (P, I) be a Datalog¬ database. Let J be a Herbrand interpretation such that $I \subseteq J$ and $J \setminus I$ consists of only atoms of intensional predicates. The *positivized ground version of P given J*, denoted by $pg(P, J)$, is the smallest set of ground Datalog program clauses such that: if $\varphi = (A \leftarrow B_1, \ldots, B_n)$ is a ground instance of a program clause from P that uses only constants occurring in P or J, and $C_i \notin J$ for all $1 \leq i \leq n$ such that $B_i = \sim C_i$ is a negative literal, then the clause obtained from φ by deleting all negative literals in its body belongs to $pg(P, J)$.

We define

$$conseq_{P,I}(J) = T_{pg(P,J)} \uparrow \omega$$

$$I_0 = conseq_{P,I}(B_{P,I})$$

$$I_{n+1} = conseq_{P,I}(I_n) \quad \text{for } n \geq 0.$$

Observe that the operator $conseq_{P,I}$ is antimonotonic (w.r.t. $\subseteq$). Hence

$$I_0 \subseteq I_2 \subseteq I_4 \subseteq \ldots \subseteq I_5 \subseteq I_3 \subseteq I_1. \tag{1}$$

Let

$$I_* = \bigcup_{i \geq 0} I_{2i} \text{ and } I^* = \bigcap_{i \geq 0} I_{2i+1}.$$

The *well-founded model* of (P, I) is the three-valued Herbrand interpretation

$$I_*^* = I_* \cup \{\sim A \mid A \notin I^*\}$$

We will denote I_n, I_*, I^*, I_*^* also by $conseq_n(P, I)$, $conseq_*(P, I)$, $conseq^*(P, I)$, $WF(P, I)$, respectively.

A *query* to a Datalog$^\neg$ database (P, I) is a formula of the form $q(\bar{t})$, where q is an intensional predicate and $\bar{t}$ is a tuple of terms. An *answer* for a query $q(\bar{t})$ to a Datalog$^\neg$ database (P, I) w.r.t. the well-founded semantics is a ground instance $\bar{t}'$ of $\bar{t}$ such that $q(\bar{t}') \in WF(P, I)$. Without loss of generality, we will consider only queries of the form $q(\bar{x})$, where $\bar{x}$ is a tuple of pairwise distinct variables. The reason is that, if $q(\bar{t})$ is a query, $\bar{x}$ is the tuple of all pairwise distinct variables in $\bar{t}$, p is a fresh intensional predicate of the same arity of $\bar{x}$ and θ is a substitution, then $\bar{t}\theta$ is an answer for the query $q(\bar{t})$ to (P, I) w.r.t. the well-founded semantics if $\bar{x}\theta$ is an answer for the query $p(\bar{x})$ to (P', I) w.r.t. the well-founded semantics, where $P' = P \cup \{p(\bar{x}) \leftarrow q(\bar{t})\}$.

Query-Subquery Nets for Datalog$^\neg$

Query-subquery nets (QSQN) and the QSQN evaluation method were defined previously (Nguyen and Cao 2012) for evaluating queries to a Horn knowledge base. They were extended to QSQN-STR by Cao (2015) for dealing with stratified knowledge bases. In this section, we make a further extension for dealing with Datalog$^\neg$ under the well-founded semantics.

Let P be a Datalog$^\neg$ program and $\varphi_1, \ldots, \varphi_m$ be all the program clauses of P, with $\varphi_i = (A_i \leftarrow B_{i,1}, \ldots, B_{i,ni})$ for $1 \leq i \leq m$.

Definition 6 (QSQN-WF Structure)

A *query-subquery net structure* of a Datalog$^\neg$ program P, denoted by QSQN-WF structure, is a tuple (V, E, T), where V is a set of nodes, E is a set of edges, and T is a function called the *memorizing type* of the net structure. In particular,

– V consists of the following nodes:
  - *input_p* and *ans_p*, for each intensional predicate p of P,
  - pre_filter_i, $filter_{i,1}$, …, $filter_{i,ni}$, $post\_filter_i$, for each $1 \leq i \leq m$.

– E consists of the following edges:
  - $(filter_{i,1}, filter_{i,2})$, …, $(filter_{i,ni-1}, filter_{i,ni})$, for each $1 \leq i \leq m$,

- ($input\_p$, pre_filter_i) and ($post\_filter_i$, ans_p), for each $1 \leq i \leq m$, where p is the predicate of A_i,
- (pre_filter_i, $filter_{i,1}$) and ($filter_{i,ni}$, $post\_filter_i$), for each $1 \leq i \leq m$ such that $n_i \geq 1$,
- (pre_filter_i, $post\_filter_i$), for each $1 \leq i \leq m$ such that $n_i = 0$,
- ($filter_{i,j}$, $input\_p$), for each intensional predicate p, $1 \leq i \leq m$ and $1 \leq j \leq n_i$ such that $B_{i,j}$ is an atom of p,
- (ans_p, $filter_{i,j}$), for each intensional predicate p, $1 \leq i \leq m$ and $1 \leq j \leq n_i$ such that $B_{i,j}$ is an atom of p and $B_{i,j}$ is a positive literal.

– T maps each $filter_{i,j} \in V$ such that the predicate of $B_{i,j}$ is extensional to *true* or *false* (As can be seen later, the aim of T is that if $T(filter_{i,j}) = false$ then subqueries for $filter_{i,j}$ are always processed immediately without being accumulated at $filter_{i,j}$).

If $(v, w) \in E$ then we call w a *successor* of v. Note that V and E are uniquely specified by P. The pair (V, E) is called the QSQN-WF topological structure of P.

If P is a stratified Datalog$^\neg$ program, then a QSQN-WF structure of P is also a QSQN-STR structure of P.

Example 1. Consider the following Datalog$^\neg$ program P (Gelder, Ross, and Schlipf 1991; Abiteboul, Hull, and Vianu 1995), where *win* is an intensional predicate, *moves* is an extensional predicate, and x, y are variables:

$$win(x) \leftarrow moves(x, y), \sim win(y).$$

Figure 1 illustrates the QSQN-WF topological structure of this program.

Example 2. This example was given in (Ramamohanarao and Harland 1994). It uses the following Datalog$^\neg$ program P:

$$path(x, y) \leftarrow edge(x, y)$$

$$path(x, y) \leftarrow edge(x, z), path(z, y)$$

$$acyclic(x, y) \leftarrow path(x, y), \sim path(y, x)$$

where *path* and *acyclic* are intensional predicates, *edge* is an extensional predicate and x, y, z are variables. The QSQN-WF topological structure of the program P is illustrated in Figure 2.

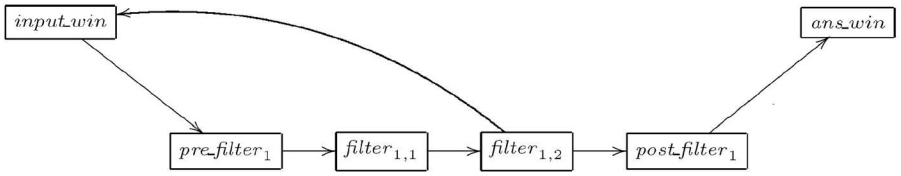

Figure 1. The QSQN-WF topological structure of the program given in Example 1.

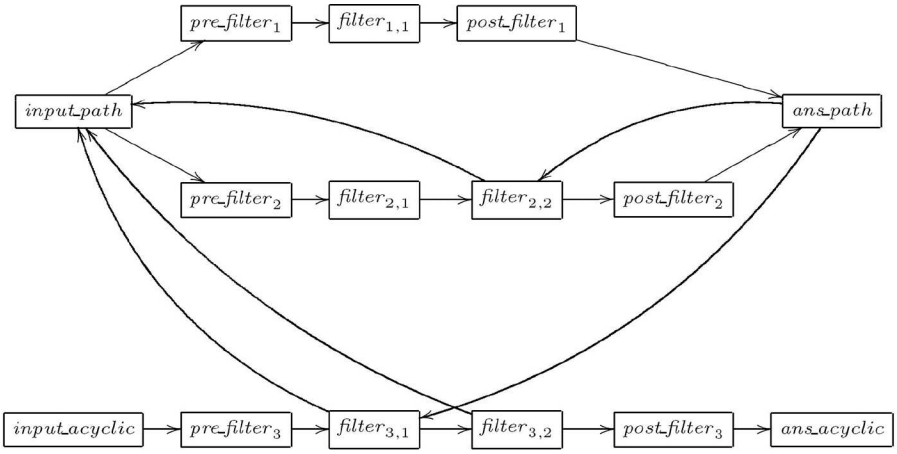

Figure 2. The QSQN-WF topological structure of the program given in Example 2.

Definition 7 (QSQN-WF)

A *query-subquery net* of a Datalog¬ program P (under the well-founded semantics), denoted by QSQN-WF, is a tuple $N = (V, E, T, C)$ such that (V, E, T) is a QSQN-WF structure of P, and C is a mapping that associates each node $v \in V$ with a structure called the *contents* of v with the following properties:

- If $v \in \{input\_p, ans\_p\}$ then $C(v)$ consists of:
  - $tuples(v)$: a set of pairs $(k, \bar{t})$, where $k \in \mathbb{N}$ and $\bar{t}$ is a tuple of terms with the same arity as p,
  - $unprocessed(v, w)$ for each $(v, w) \in E$: a subset of $tuples(v)$.
- If $v = pre\_filter_i$ then $C(v)$ consists of:
  - $atom(v) = A_i$,
  - $post\_vars(v) = Vars((B_{i,1}, \ldots, B_{i,ni}))$,
  - $pos\_clause(v) = true$ if all $B_{i,1}, \ldots, B_{i,ni}$ are positive literals, and $pos\_clause(v) = false$ otherwise.
- If $v = post\_filter_i$ then $C(v)$ is empty, but we assume $pre\_vars(v) = \emptyset$,
- If $v = filter_{i,j}$ and p is the predicate of $B_{i,j}$ then $C(v)$ consists of:
  - $neg(v) = true$ if $B_{i,j}$ is a negative atom, and $neg(v) = false$ otherwise,
  - $kind(v) = extensional$ if p is extensional, and $kind(v) = intensional$ otherwise,
  - $pred(v) = p$ (called the predicate of v) ,
  - $atom(v) = B_{i,j}$ if $B_{i,j}$ is a positive literal, and $atom(v) = B'$ if $B_{i,j} = \sim B'$,
  - $pre\_vars(v) = Vars((B_{i,j}, \ldots, B_{i,ni}))$ and $post\_vars(v) = Vars((B_{i,j+1}, \ldots, B_{i,ni}))$,
  - $subqueries(v)$: a set of tuples of the form $(k, \bar{t}, \delta)$, where $k \in \mathbb{N}$, $\bar{t}$ is a tuple of terms with the same arity as the predicate of A_i, and δ is an idempotent substitution such that $dom(\delta) \subseteq pre\_vars(v)$ and $dom(\delta) \cap Vars(\bar{t}) = \emptyset$,
  - $unprocessed\_subqueries(v) \subseteq subqueries(v)$,

- in the case p is intensional:
  * $unprocessed\_subqueries_2(v) \subseteq subqueries(v)$,
- in the case p is intensional and $neg(v) = false$:
  * *unprocessed_tuples(v):* a set of pairs $(k, \bar{t})$, where $k \in \mathbb{N}$ and $\bar{t}$ is a tuple of terms with the same arity as p.

– If $v = filter_{i,j}$, $kind(v) = extensional$ and $T(v) = false$, then $subqueries(v)$ and $unprocessed\_subqueries(v)$ are empty (and we can ignore them).

Observe that, for each $(v, w) \in E$:

– if v is either pre_filter_i or $post\_filter_i$ or $filter_{i,j}$ with $kind(v) = extensional$, then v has exactly one successor, denoted by $succ(v)$;

– if v is $filter_{i,j}$ with $kind(v) = intensional$ and $pred(v) = p$, then v has exactly two successors: $succ(v) = filter_{i,j+1}$ if $n_i > j$; $succ(v) = post\_filter_i$ otherwise; and $succ_2(v) = input\_p$.

Algorithm 1. Evaluating a Datalog¬ query under the well-founded semantics.

Input: a Datalog¬ database (P, I) and a query $q(\bar{x})$.
Output: all tuples $\bar{t}$ such that $q(\bar{t})$ is in $WF(P, I)$.

```
let (V, E, T) be a QSQN-WF structure of P;
// T can be chosen arbitrarily or appropriately
set C so that N = (V, E, T, C) is an empty QSQN-WF of P;
h := 0, n := −2;
repeat
    n := n + 2;
    let x̄′ be a fresh variant of x̄;
    add the pair (n, x̄′) to tuples(input_q);
    foreach (input_q, v) ∈ E do
        if n > 0 or pos_clause(v) then
            add the pair (n, x̄′) to unprocessed(input_q, v);
    while there exists (u, v) ∈ E such that active-edge(u, v) holds do
        select such an edge (u, v);          // any strategy can be used
        fire(u, v);
    if all_answers(n) ≠ all_answers(h) or all_answers(h) was extended during the
    current iteration of the "repeat" loop then h := n;
    else if all_inputs(h) was not extended during the current iteration of the "repeat"
    loop then break;
until false;
return all_tuples(ans_q, n);
```

Example 3. Figure 3 illustrates a QSQN-WF of the program P given in Example 1.

A QSQN-WF of P is *empty* if all the sets of the form $tuples(v)$, $unprocessed(v, w)$, $subqueries(v)$, $unprocessed\_subqueries(v)$, $unprocessed\_subqueries_2(v)$,

$$win(x) \leftarrow moves(x, y), \sim win(y).$$

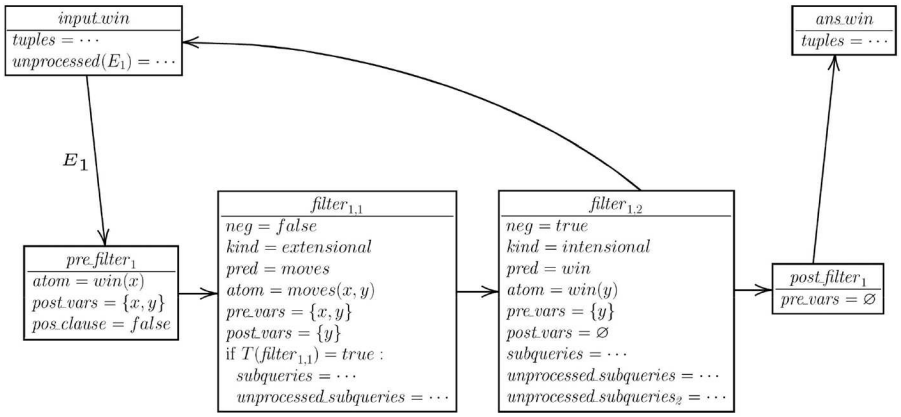

Figure 3. The QSQN-WF of the program given in Example 1.

and *unprocessed_tuples*(v) are empty. Assume that (P, I) is the considered Datalog¬ database. An index $k \in \mathbb{N}$ in tuples from those sets are related to the computation of $I_k = conseq_k(P, I)$. QSQN-WF differs from QSQN-STR (Cao 2015) by having those indices and the attribute *pos_clause*(v) for $v = pre\_filter_i$. We denote

tuples(v, k) = $\{\bar{t} \mid (k, \bar{t}) \in tuples(v)\}$

all_inputs(k) = $\bigcup\{p(\bar{t}) \mid p$ is an intensional predicate and $\bar{t} \in tuples(input\_p, k)\}$.

Because of (1), for a natural number k, we also define:

all_tuples(*ans_p*, $2k$) = $\bigcup_{h=0}^{k} tuples(ans\_p, 2h)$

all_tuples(*ans_p*, $2k+1$) = *all_tuples*(*ans_p*, $2k$) ∪ *tuples*(*ans_p*, $2k+1$)

all_answers(k) = $\bigcup\{p(\bar{t}) \mid p$ is an intensional predicate and $\bar{t} \in tuples(ans\_p, k)\}$.

Let $\bar{t}$ and $\bar{t}'$ be tuples of terms. We say that $\bar{t}$ is *more general* than $\bar{t}'$, and $\bar{t}'$ is an *instance* of $\bar{t}$, if there exists a substitution θ such that $\bar{t}\theta = \bar{t}'$. We say that a pair (k, $\bar{t}$) is *more general* than (k', $\bar{t}'$), and (k', $\bar{t}'$) is *less general* than (k, $\bar{t}$), if $k = k'$ and $\bar{t}'$ is an instance of $\bar{t}$.

A *subquery* is a tuple of the form (k, $\bar{t}$, δ), where $k \in \mathbb{N}$, $\bar{t}$ is a tuple of terms, and δ is an idempotent substitution such that $dom(\delta) \cap Vars(\bar{t}) = \emptyset$. The set *unprocessed_subqueries*$_2$(v) (resp. *unprocessed_subqueries*(v)) contains the subqueries that were not transferred through the edge (v, *succ*$_2$(v)) (resp. (v, *succ*(v))). We say that (k, $\bar{t}$, δ) is *more general* than (k', $\bar{t}'$, δ') w.r.t. v, and (k', $\bar{t}'$, δ') is *less general* than (k, $\bar{t}$, δ) w.r.t. v, if $k = k'$ and there exists a substitution θ such that $\bar{t}\theta = \bar{t}'$ and $(\delta\theta)_{|pre\_vars(v)} = \delta'$.

A QSQN-WF is said to be *stable at level k* if it does not contain any pair (k, $\bar{t}$) in any set of the form *unprocessed*(v, w) or *unprocessed_tuples*(v), and any tuple (k, $\bar{t}$, δ) in any set of the form *unprocessed_subqueries*(v) or *unprocessed_subqueries*$_2$(v).

Algorithm 1 presents our method for evaluating a Datalog¬ query under the well-founded semantics. It is based on SLS-resolution (Przymusinski 1989) and

the alternating fixpoint characterization (Gelder 1993). During each iteration of the main loop, which is used to reach the alternating fixpoint, the algorithm selects an active edge and fires the operation for the edge. The function **active-edge**(*u, v*) in the appendix returns *true* if some data accumulated in *u* can be processed to produce data to transfer through the edge (*u, v*). If the function **active-edge**(*u, v*) returns *true* then the procedure **fire**(*u, v*) in the appendix processes the data accumulated in *u* that has not been processed before and transfers appropriate data through the edge (*u, v*). The procedure **fire** uses the procedures **add-tuple, add-subquery** and **transfer** in the appendix. The procedure **transfer**(*D, u, v*) specifies the effects of transferring data *D* through the edge (*u, v*) of a QSQN-WF.

Due to the lack of space, a trace of running Algorithm 1 step by step on an example query together with an intuitive presentation is provided online (Cao 2016b).

Data Complexity, Soundness, and Completeness

The *data complexity* of an algorithm that evaluates a query $q(\bar{x})$ to a deductive database (P, I) is measured in the size of the extensional instance I, while assuming that the used predicates, the program P and the query $q(\bar{x})$, are fixed.

Lemma 1. *Algorithm 1 terminates and has PTIME data complexity.*

Proof. (*Sketch*). Let m be the size of I. Observe that $|B_{P,I}| \leq f(m)$ for some polynomial f. During the run of Algorithm 1, $all\_answers(h) \subseteq B_{P,I}$ and thus $|all\_answers(h)| \leq f(m)$. As relations $tuples(input\_p)$ keep only fresh variants of the most general pairs, we also have that $|all\_inputs(h)| \leq g(m)$ for some polynomial g. Each iteration except the last one of the "repeat" loop extends either $all\_answers(h)$ or $all\_inputs(h)$, where $all\_answers(h)$ is accumulative when h increases. Hence, the loop repeats no more than $f(m) \times g(m)$ times. Observe that each iteration of that loop is executed in polynomial time in m. Therefore, Algorithm 1 terminates and has PTIME data complexity.

Lemma 2. *Let (P, I) be a Datalog$^\neg$ database. During the run of Algorithm 1 for (P, I) and a query, for every intensional predicate p of P, every tuples $\bar{t}$, $\bar{t}'$ and every natural number k:*

1. *if $p(\bar{t}) \in all\_answers(k)$, then $p(\bar{t}) \in conseq_k(P, I)$,*
2. *if $p(\bar{t}) \in all\_inputs(k)$, the net N is stable at the level k, $p(\bar{t}') \in conseq_k(P, I)$ and $\bar{t}'$ is an instance of $\bar{t}$, then $p(\bar{t}') \in all\_answers(k)$.*

The first assertion of the above lemma states soundness of Algorithm 1, while the second one states conditional completeness of Algorithm 1. This lemma can be proved in a similar way as done for Lemma 5.1 in (Cao 2016a).

Lemma 3. *Let h_0 and n_0 be the values of h and n, respectively, when Algorithm 1 terminates. Then:*

1. $0 \leq h_0 \leq n_0 - 2$, *all_answers*(h_0) = *all_answers*(n_0), *all_answers*(h_0), and *all_inputs*(h_0) were not changed during the last iteration of the "repeat" loop,
2. *for every intensional predicate p, every* $\bar{t} \in$ *tuples(input_p,* h_0*) is an instance of a tuple from tuples(input_p,* $h_0 + 2$*).*

Proof. The first assertion clearly holds. Consider the second assertion. Suppose that $\bar{t} \in$ *tuples(input_p,* h_0*)* and *(*h_0*,* $\bar{t}$*)* was added to *tuples(input_p)* due to the processing of a pair *(k,* $\bar{x}'$*)* in *tuples(input_q)* with an even $k \geq h_0$. Since *all_inputs*(h_0) was not changed during the last iteration of the "repeat" loop, $k \leq n_0$–2. There exists *(*$k + 2$*,* $\bar{x}''$*)* $\in$ *tuples(input_q)*, where $\bar{x}''$ is a tuple of pairwise distinct variables. The processing of *(*$k + 2$*,* $\bar{x}''$*)* in *tuples(input_q)* was similar to the processing of *(k,* $\bar{x}'$*)* in *tuples(input_q)* and caused addition of *(*$h_0 + 2$*,* $\bar{t}'$*)* to *tuples(input_p)* for some tuple $\bar{t}'$ more general than $\bar{t}$. The reason is that at any moment after the h_0^{th} iteration of the "repeat" loop, for any even number l between h_0 and $n_0 - 2$, *all_answers(l)* = *all_answers*(h_0) and $conseq_{P,I}$(*all_answers(l)* ∪ *I)* = $conseq_{P,I}$(*all_answers*(h_0) ∪ *I)*.

Lemma 4. *If the "repeat" loop of Algorithm 1 is allowed to run forever by deleting the steps 14 and 15, then no further pair (k,* $\bar{t}$*) with an even k will be added to any relation of the form tuples(ans_p).*

Proof. (*Sketch*). Let h_0 and n_0 be the values of h and n, respectively, when Algorithm 1 terminates. We have that $h_0 + 2 \leq n_0$. Now, assume that the loop is allowed to run forever by deleting the steps 14 and 15. We first show that:

$$\begin{aligned} &\text{for any pair } (k, \bar{t}) \text{ in any relation } tuples(input\_p) \text{ with an even} \\ &k > h_0 + 2, \text{there exists } (h_0 + 2,\ \bar{t}') \in\ tuples(input\_p) \text{ such that} \\ &\bar{t} \text{ is an instance of } \bar{t}' \text{ and } (h_0 + 2,\ \bar{t}') \text{ was added to } tuples(input\_p) \\ &\text{no later than the } n_0^{th} \text{ iteration of the "repeat" loop.} \end{aligned} \tag{2}$$

We prove this by induction on the moment when $(k, \bar{t})$ is added to *tuples* (*input_p*). The case when that addition occurs at the step 7 of Algorithm 1 is clear. Consider the case when that addition occurs at the step 16 of the procedure **transfer**. It must originate from the processing of a pair $(k + 2, \bar{s})$ from *tuples*(*input_p*). By the inductive assumption of (2), there exists a pair $(h_0 + 2, \bar{s}') \in$ *tuples*(*input_p*) such that $\bar{s}$ is an instance of $\bar{s}'$ and $(h_0 + 2, \bar{s}')$ was added to *tuples*(*input_p*) no later than the n_0^{th} iteration of the "repeat" loop. The processing of $(h_0 + 2, \bar{s}')$ in *tuples*(*input_p*) must have added a pair $(h_0, \bar{t}'')$ to *tuples*(*input_p*) for some $\bar{t}''$ more general than $\bar{t}$. The reason is that *all_answers*(h_0) = *all_answers*(k) and $conseq_{P,I}$(*all_answers*(h_0) ∪ *I*) = $conseq_{P,I}$(*all_answers*(k) ∪ *I*). By the second assertion of Lemma 3, at the end of the n_0^{th} iteration of the "repeat" loop, there exists $(h_0 + 2, \bar{t}') \in$ *tuples*(*input_p*) such that $\bar{t}''$ is an instance of $\bar{t}'$. Clearly, $\bar{t}$ is an instance of $\bar{t}'$.

Since Algorithm 1 keeps in relations *tuples*(*input_p*) only the most general pairs, as a consequence of (2), we have that:

$$\begin{aligned} &\text{for any } k \leq h_0 + 2,\ \mathit{all\_inputs}(k) \text{ and } \mathit{all\_answers}(k) \\ &\quad \text{do not change after the } n_0^{th} \text{ iteration of the ``repeat'' loop.} \end{aligned} \tag{3}$$

Now, let us consider the assertion of the lemma. By (2), the processing of any pair $(k', \bar{t}')$ in *tuples*(*input_p*) with an even $k' > h_0 + 2$ is subsumed by the processing of a pair $(h_0 + 2, \bar{t}'')$ in *tuples*(*input_p*) with $\bar{t}''$ being more general than $\bar{t}'$. Any candidate pair $(k, \bar{t})$ that would be added to *tuples*(*ans_p*) by the former has earlier been added to *tuples*(*ans_p*) by the latter. Once again, the reason is that $\mathit{all\_answers}(h_0) = \mathit{all\_answers}(k' - 2)$ and $\mathit{conseq}_{P,I}(\mathit{all\_answers}(h_0) \cup I) = \mathit{conseq}_{P,I}(\mathit{all\_answers}(k' - 2) \cup I)$. By the first assertion of Lemma 3, subqueries of the form $(h_0 + 2, \bar{t}'')$ in *tuples*(*input_p*) do not cause addition of any pair $(k, \bar{t})$ with an even k to *tuples*(*ans_p*) during the n_0^{th} iteration of the "repeat" loop. By (3), it follows that no pair $(k, \bar{t})$ with an even k will be added to any relation of the form *tuples*(*ans_p*) after the n_0^{th} iteration of the "repeat" loop.

Theorem 1. *Algorithm 1 is correct and has PTIME data complexity.*

Proof. By Lemma 1, Algorithm 1 has PTIME data complexity. As the soundness, by the assertion 1 of Lemma 2, for every tuple $\bar{t}$ returned by Algorithm 1, $q(\bar{t}) \in WF(P, I)$. For the completeness, let $q(\bar{t}) \in WF(P, I)$, i.e., $q(\bar{t}) \in \mathit{conseq}_k(P, I)$ for some even number k. We show that $\bar{t} \in \mathit{all\_tuples}(\mathit{ans\_q}, k)$. By Lemma 4, without loss of generality, we assume that the "repeat" loop is allowed to run forever by deleting the steps 14 and 15. Then, since $(k, \bar{x}') \in \mathit{tuples}(\mathit{input\_q})$ and the net N is stable at the level k after the k^{th} iteration of that "repeat" loop and $\bar{t}$ is an instance of $\bar{x}'$, by the assertion 2 of Lemma 2, $\bar{t} \in \mathit{all\_tuples}(\mathit{ans\_q}, k)$. This completes the proof.

Conclusions

We have developed the QSQN-WF method as the first one for evaluating queries to Datalog$^{\neg}$ databases under the well-founded semantics that is set-at-a-time and strictly goal-directed w.r.t. SLS-resolution. These properties are important for reducing accesses to the secondary storage and redundant computations. Our method follows SLS-resolution, with Van Gelder's alternating fixpoint semantics on the background, but uses a query-subquery net to implement tabulation and the set-at-a-time technique, reduce redundant computations, and allow any control strategy within each iteration of the main loop. We have proved that the QSQN-WF method is sound and complete w.r.t. the well-founded semantics and has PTIME data complexity. As future work, we intend to develop an appropriate control strategy for QSQN-WF, implement the method, and compare it empirically with the other ones.

Another area of interest for the application of the proposed method is in working with large datasets (Tachmazidis, Antoniou, and Faber 2014; Vossen 2014).

Funding

This work was supported by the Polish National Science Centre (NCN) under Grant No. 2011/02/A/HS1/00395.

References

Abiteboul, S., R. Hull, and V. Vianu. 1995. *Foundations of databases*. Boston, MA: Addison Wesley.

Beeri, C., and R. Ramakrishnan. 1991. On the power of magic. *Journal of Logic Programming* 10:255–99.

Cao, S. T. 2015. Query-subquery nets with stratified negation. *Proceedings of ICCSAMA'2015, Advances in Intelligent Systems and Computing* 358:355–66.

Cao, S. T. 2016a. Methods for evaluating queries to Horn knowledge bases in first-order logic. PhD dissertation. University of Warsaw, Warsaw, Poland. Available at http://mimuw.edu.pl/~sonct/stc-thesis.pdf

Cao, S. T. 2016b. An illustration of Algorithm 1 on an example. Available at http://mimuw.edu.pl/~sonct/Alg1.zip

Cao, S. T., and L. A. Nguyen. 2015. An empirical approach to query-subquery nets with tail-recursion elimination. In *New Trends in Database and Information Systems II, selected papers of ADBIS'2014, Advances in Intelligent Systems and Computing* 312:109–20.

Chen, W., T. Swift, and D. S. Warren. 1995. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming* 24 (3):161–99. doi:10.1016/0743-1066(94)00028-5

Drabent, W. 1995. What is failure? An approach to constructive negation. *Acta Informatica* 32 (1):27–59. doi:10.1007/bf01185404

Gelder, A. V. 1993. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences* 47 (1):185–221. doi:10.1016/0022-0000(93)90024-q

Gelder, A. V., K. A. Ross, and J. S. Schlipf. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 38 (3):619–49. doi:10.1145/116825.116838

Kemp, D. B., D. Srivastava, and P. J. Stuckey. 1995. Bottom-up evaluation and query optimization of well-founded models. *Theoretical Computer Science* 146 (1 & 2):145–84. doi:10.1016/0304-3975(94)00153-a

Lloyd, J. W. 1987. *Foundations of logic programming*. 2nd ed. Berlin: Springer.

Madalinska-Bugaj, E., and L. A. Nguyen. 2012. A generalized QSQR evaluation method for horn knowledge bases. *ACM Transactions on Computational Logic* 13 (4):1–28. doi:10.1145/2362355.2362360

Morishita, S. 1996. An extension of Van Gelder's alternating fixpoint to magic programs. *Journal of Computer and System Sciences* 52 (3):506–21. doi:10.1006/jcss.1996.0038

Nguyen, L. A., and S. T. Cao. 2012. Query-subquery nets. Proceedings of ICCCI'2012, Lecture Notes in Computer Science 7635:239–48.

Przymusinski, T. C. 1989. Every logic program has a natural stratification and an iterated least fixed point model. In *Proceedings of PODS'1989*, 11–21, ACM.

Ramamohanarao, K., and J. Harland. 1994. An introduction to deductive database languages and systems. *The VLDB Journal* 3 (2):107–22. doi:10.1007/bf01228878

Ross, K. A. 1992. A procedural semantics for well-founded negation in logic programs. *Journal of Logic Programming* 13 (1):1–22. doi:10.1016/0743-1066(92)90019-y

Tachmazidis, I., G. Antoniou, and W. Faber. 2014. Efficient computation of the well-founded semantics over big data. *Theory and Practice of Logic Programming* 14:445–59. doi:10.1017/s1471068414000131

Vieille, L. 1989. Recursive query processing: The power of logic. *Theoretical Computer Science* 69 (1):1–53. doi:10.1016/0304-3975(89)90088-1

Vossen, G. 2014. Big data as the new enabler in business and other intelligence. *Vietnam Journal of Computer Science* 1 (1):3–13. doi:10.1007/s40595-013-0001-6

Appendix A

Functions and Procedures Used for Algorithm 1

This appendix presents a list of all functions and procedures that are used for Algorithm 1.

Function active-edge(u, v)

Global data: a QSQN-WF $N = (V, E, T, C)$.
Input: an edge $(u, v) \in E$.
Output: *true* if there are some data to transfer through the edge (u, v), and *false* otherwise.

if u *is* pre_filter_i *or* $post\_filter_i$ **then return** *false*;
else if u *is* $input\_p$ *or* ans_p **then return** $unprocessed(u, v) \neq \emptyset$;
else if u *is* $filter_{i,j}$ *and* $kind(u) = extensional$ **then**
    return $T(u) = true \wedge unprocessed\_subqueries(u) \neq \emptyset$;
else // u is of the form $filter_{i,j}$ and $kind(u) = intensional$
    let $filter_{i,j} = u$ and $p = pred(u)$;
    if $v = input\_p$ **then return** $unprocessed\_subqueries_2(u) \neq \emptyset$;
    else if $neg(u) = false$ **then**
        return $unprocessed\_subqueries(u) \neq \emptyset \vee unprocessed\_tuples(u) \neq \emptyset$;
    else
        if *there exists* $(k, \bar{t}, \delta) \in unprocessed\_subqueries(u)$ *such that* $atom(u)\delta \in \{p(\bar{t}') \mid \bar{t}' \in all\_tuples(ans\_p, k-1)\}$ **then return** *true*;
        if $\neg$**active-edge**$(u, input\_p)$ *and there exists* $(k, \bar{t}, \delta) \in unprocessed\_subqueries(u)$ *such that the net N is stable at the level* $k-1$ **then return** *true*;
        else return *false*;

Procedure add-subquery$(k, \bar{t}, \delta, \Gamma, v)$

Purpose: add the subquery $(k, \bar{t}, \delta)$ to Γ, but keep in Γ only the most general subqueries w.r.t. v.
if *no subquery in* Γ *is more general than* $(k, \bar{t}, \delta)$ *w.r.t.* v **then**
    delete from Γ all subqueries less general than $(k, \bar{t}, \delta)$ w.r.t. v;
    add $(k, \bar{t}, \delta)$ to Γ;

Procedure add-tuple$(k, \bar{t}, \Gamma)$

Purpose: add the pair $(k, \bar{t})$ to Γ, but keep in Γ only the most general pairs.
let $\bar{t}'$ be a fresh variant of $\bar{t}$;
if *no pair from* Γ *is more general than* $(k, \bar{t}')$ **then**
    delete from Γ all pairs that are less general than $(k, \bar{t}')$;
    add $(k, \bar{t}')$ to Γ;

Procedure **fire**(u, v)

Global data: a Datalog$^{\neg}$ program P, an extensional instance I, a QSQN-WF $N = (V, E, T, C)$ of P.
Input: an edge $(u, v) \in E$ such that **active-edge**(u, v) holds.

if u *is input_p or ans_p* **then**
  if u *is input_p* **then**
    delete from *unprocessed*(u, v) all pairs $(k, \bar{t})$ such that $\bar{t} \in$ *all_tuples*(ans_p, k);
  transfer(*unprocessed*(u, v), u, v);
  unprocessed$(u, v) := \emptyset$;
else if u *is* $filter_{i,j}$ *and kind*$(u) =$ *extensional and* $T(u) = true$ **then**
  let $p = pred(u)$ and set $\Gamma := \emptyset$;
  foreach $(k, \bar{t}, \delta) \in$ *unprocessed_subqueries*(u) **do**
    if $neg(u) = false$ **then**
      foreach $\bar{t}' \in I(p)$ **do**
        if $atom(u)\delta$ *is unifiable with a fresh variant of* $p(\bar{t}')$ *by an mgu* γ **then**
          add-subquery$(k, \bar{t}\gamma, (\delta\gamma)_{|post\_vars(u)}, \Gamma, v)$;
    else if $atom(u)\delta \notin \{p(\bar{t}') \mid \bar{t}' \in I(p)\}$ **then add-subquery**$(k, \bar{t}, \delta_{|post\_vars(u)}, \Gamma, v)$;
  unprocessed_subqueries$(u) := \emptyset$;
  transfer(Γ, u, v);
else if u *is* $filter_{i,j}$ *and kind*$(u) =$ *intensional* **then**
  let $p = pred(u)$ and set $\Gamma := \emptyset$;
  if $v = input\_p$ **then**
    foreach $(k, \bar{t}, \delta) \in unprocessed\_subqueries_2(u)$ **do**
      let $p(\bar{t}') = atom(u)\delta$;
      if $neg(u) = false$ **then add-tuple**$(k, \bar{t}', \Gamma)$;
      else add-tuple$(k - 1, \bar{t}', \Gamma)$;
    $unprocessed\_subqueries_2(u) := \emptyset$;
  else if $neg(u) =$ *false* **then**
    foreach $(k, \bar{t}, \delta) \in$ *unprocessed_subqueries*(u) **do**
      foreach $\bar{t}' \in$ *all_tuples*(ans_p, k) **do**
        if $atom(u)\delta$ *is unifiable with a fresh variant of* $p(\bar{t}')$ *by an mgu* γ **then**
          add-subquery$(k, \bar{t}\gamma, (\delta\gamma)_{|post\_vars(u)}, \Gamma, v)$;
    unprocessed_subqueries$(u) := \emptyset$;
    foreach $(k, \bar{t}) \in$ *unprocessed_tuples*(u) **do**
      foreach $(k', \bar{t}', \delta) \in$ *subqueries*(u) **do**
        if ($k = k'$ *or* k *is even and less than* k') *and* ($atom(u)\delta$ *is unifiable with a fresh variant of* $p(\bar{t})$ *by an mgu* γ) **then**
          add-subquery$(k', \bar{t}'\gamma, (\delta\gamma)_{|post\_vars(u)}, \Gamma, v)$;
    unprocessed_tuples$(u) := \emptyset$;
  else
    foreach $(k, \bar{t}, \delta) \in$ *unprocessed_subqueries*(u) **do**
      if $atom(u)\delta \in \{p(\bar{t}') \mid \bar{t}' \in$ *all_tuples*$(ans\_p, k - 1)\}$ **then**
        delete the tuple $(k, \bar{t}, \delta)$ from *unprocessed_subqueries*(u);
      else if $\neg$**active-edge**$(u, input\_p)$ *and the net N is stable at the level* $k - 1$ **then**
        add-subquery$(k, \bar{t}, \delta_{|post\_vars(u)}, \Gamma, v)$;
        delete the tuple $(k, \bar{t}, \delta)$ from *unprocessed_subqueries*(u);
  transfer(Γ, u, v);

Procedure transfer(D, u, v)

Global data: a Datalog$^\neg$ program, an extensional instance I, a QSQN-WF $N = (V, E, T, C)$ of P.
Input: data D to transfer through the edge $(u, v) \in E$.

if $D = \emptyset$ **then return;**
if u *is input_p* **then**
  $\Gamma := \emptyset$;
  foreach $(k, \bar{t}) \in D$ **do**
    if $p(\bar{t})$ *and atom*(v) *are unifiable by an mgu* γ **then**
      add-subquery$(k, \bar{t}\gamma, \gamma_{|post\_vars(v)}, \Gamma, succ(v))$;
  transfer$(\Gamma, v, succ(v))$;
else if u *is ans_p* **then** $unprocessed\_tuples(v) := unprocessed\_tuples(v) \cup D$;
else if v *is input_p* **then**
  foreach $(k, \bar{t}) \in D$ **do**
    let $\bar{t}'$ be a fresh variant of $\bar{t}$;
    if $(k, \bar{t}')$ *is not less general than any pair from* $tuples(v)$ **then**
      foreach $(k, \bar{t}'') \in tuples(v)$ *that is less general than* $(k, \bar{t}')$ **do**
        delete $(k, \bar{t}'')$ from $tuples(v)$;
        foreach $(v, w) \in E$ **do**
          delete $(k, \bar{t}'')$ from $unprocessed(v, w)$;
      add $(k, \bar{t}')$ to $tuples(v)$;
      foreach $(v, w) \in E$ **do**
        if $k > 0$ *or pos_clause*(w) **then**
          add $(k, \bar{t}')$ to $unprocessed(v, w)$;
else if v *is ans_p* **then**
  foreach $(k, \bar{t}) \in D$ *such that* $\bar{t} \notin all\_tuples(v, k)$ **do**
    if k *is even* **then**
      foreach $(k', \bar{t}) \in tuples(v)$ *such that* $k' > k$ **do**
        delete $(k', \bar{t})$ from $tuples(v)$;
        foreach $(v, w) \in E$ **do** delete $(k', \bar{t})$ from $unprocessed(v, w)$;
    add $(k, \bar{t})$ to $tuples(v)$;
    foreach $(v, w) \in E$ **do** add $(k, \bar{t})$ to $unprocessed(v, w)$;
else if v *is* $filter_{i,j}$ *and* $kind(v) = extensional$ *and* $T(v) = false$ **then**
  let $p = pred(v)$ and set $\Gamma := \emptyset$;
  foreach $(k, \bar{t}, \delta) \in D$ **do**
    if $neg(v) = false$ **then**
      foreach $\bar{t}' \in I(p)$ **do**
        if $atom(v)\delta$ *is unifiable with a fresh variant of* $p(\bar{t}')$ *by an mgu* γ **then**
          add-subquery$(k, \bar{t}\gamma, (\delta\gamma)_{|post\_vars(v)}, \Gamma, succ(v))$;
    else if $atom(v)\delta \notin \{p(\bar{t}') \mid \bar{t}' \in I(p)\}$ **then**
      add-subquery$(k, \bar{t}, \delta_{|post\_vars(v)}, \Gamma, succ(v))$;
  transfer$(\Gamma, v, succ(v))$;
else if v *is* $filter_{i,j}$ *and* ($kind(v) = extensional$ *and* $T(v) = true$ *or* $kind(v) = intensional$) **then**
  foreach $(k, \bar{t}, \delta) \in D$ **do**
    if *no subquery in* $subqueries(v)$ *is more general than* $(k, \bar{t}, \delta)$ **then**
      delete from $subqueries(v)$ and $unprocessed\_subqueries(v)$ all subqueries less general than $(k, \bar{t}, \delta)$;
      add $(k, \bar{t}, \delta)$ to both $subqueries(v)$ and $unprocessed\_subqueries(v)$;
      if $kind(v) = intensional$ **then**
        delete from $unprocessed\_subqueries_2(v)$ all subqueries less general than $(k, \bar{t}, \delta)$; add $(k, \bar{t}, \delta)$ to $unprocessed\_subqueries_2(v)$;
else // v is of the form $post\_filter_i$
  $\Gamma := \{(k, \bar{t}) \mid (k, \bar{t}, \varepsilon) \in D\}$;
  transfer$(\Gamma, v, succ(v))$;