Online Appendix

for the Paper "Incorporating Stratified Negation into Query-Subquery Nets for Evaluating Queries to Stratified Deductive Databases" [3]

This appendix contains:
– an example illustrating the QSQN-STR method (Section A),
– proofs of data complexity, soundness and completeness of the QSQN-STR method (Section B),
– experimental results (Section C), and
– the pseudocode of the QSQN-STR method (Section D).

A An Illustrative Example

The aim of this example is to illustrate how Algorithm 1 works step by step. It uses the stratified Datalog$^\neg$ database (P, I) and the stratification $P = P_1 \cup P_2$ given in [3, Example 2.2]. The QSQN-STR topological structure of the program P is illustrated in Figure 1.

$$path(x,y) \leftarrow edge(x,y)$$
$$path(x,y) \leftarrow path(x,z), edge(z,y)$$
$$acyclic(x,y) \leftarrow path(x,y), \sim path(y,x).$$

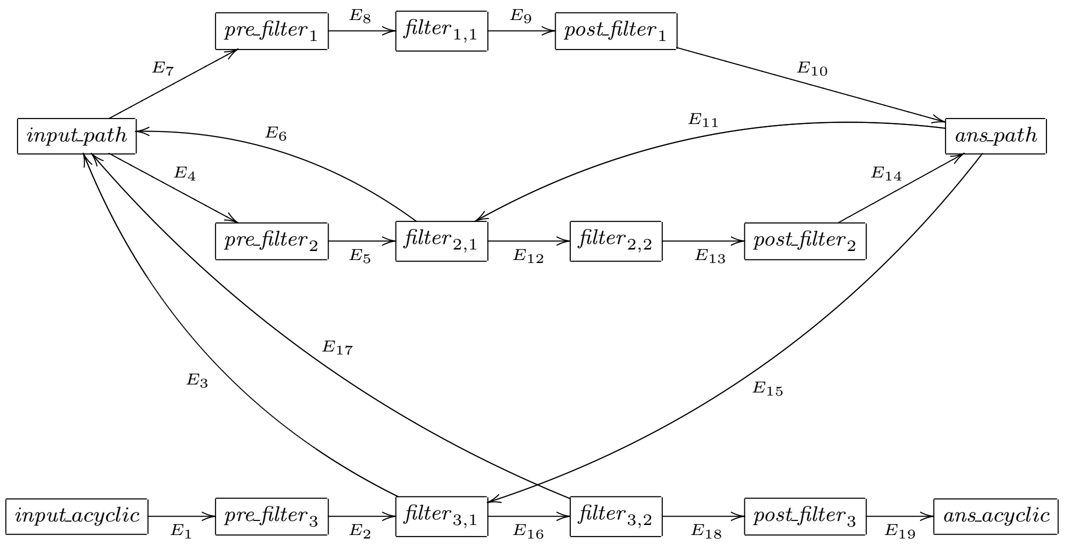
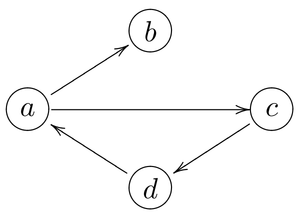


Fig. 1. The QSQN-STR topological structure of the program given in [3, Example 2.2]

Recall that the extensional instance I is specified by $I(edge) = \{(a,b),(a,c),(c,d),(d,a)\}$ and illustrated below:



We give below a trace of running Algorithm 1 for the query $acyclic(x,y)$ on the stratified Datalog$^\neg$ database (P, I). For convenience, we name the edges of the net by E_i with $1 \le i \le 19$, as shown in Figure 1. Let $T(v) = \mathit{false}$ for each $v = \mathit{filter}_{i,j}$ with $\mathit{kind}(v) = \mathit{extensional}$. Assume that Algorithm 1 evaluates the query $\mathit{acyclic}(x, y)$ to the knowledge base (P, I) using a control strategy that selects active edges for "firing" in the order (E_1, E_3, E_4, E_6, E_7, E_{11}, E_{12}, E_{11}, E_{12}, E_{11}, E_{12}, E_{15}, E_{16}, E_{17}, E_{18}), which is admissible w.r.t. strata's stability and corresponds to the IDFS2 control strategy specified in [1]. Recall that, by [3, Example 2.2], the set of answers should be $\{(a, b), (c, b), (d, b)\}$. To ease the checking, the reader can use the full pseudocode of Algorithm 1 gathered in Section D. In addition, a much more friendly presentation of this example in the PowerPoint-like mode is available online [4].

Algorithm 1 starts with an empty QSQN-STR and then adds a fresh variant (x_1, y_1) of (x, y) to the empty sets $\mathit{tuples}(\mathit{input\_acyclic})$ and $\mathit{unprocessed}(E_1)$. This makes the edge E_1 active.

1. **$\mathbf{E_1}$–$\mathbf{E_2}$**

   Firing the active edge E_1 (by `fire′`), the algorithm transfers (x_1, y_1) through this edge and empties the set $\mathit{unprocessed}(E_1)$. This produces $\{((x_1, y_1), \{x/x_1, y/y_1\})\}$ (by `transfer′`$_3$ via `transfer′`), which is then transferred through the edge E_2 and added to the empty sets $\mathit{subqueries}(\mathit{filter}_{3,1})$, $\mathit{unprocessed\_subqueries}(\mathit{filter}_{3,1})$ and $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{3,1})$ (by `transfer`$_5$ via `transfer′`).

2. **$\mathbf{E_3}$**

   Firing the active edge E_3 (by `fire′`$_1$ via `fire′`), the algorithm empties the set $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{3,1})$ and transfers (x_2, y_2), a fresh variant of (x_1, y_1), through the edge E_3 and adds its fresh variant (x_3, y_3) to the empty sets $\mathit{tuples}(\mathit{input\_path})$, $\mathit{unprocessed}(E_4)$ and $\mathit{unprocessed}(E_7)$ (by `transfer`$_2$ via `transfer′`).

3. **$\mathbf{E_4}$–$\mathbf{E_5}$**

   Firing the active edge E_4 (by `fire′`), the algorithm transfers (x_3, y_3) through this edge and empties the set $\mathit{unprocessed}(E_4)$. This produces $\{((x_3, y_3), \{x/x_3, y/y_3\})\}$ (by `transfer′`$_3$ via `transfer′`), which is then transferred through the edge E_5 and added to the empty sets $\mathit{subqueries}(\mathit{filter}_{2,1})$, $\mathit{unprocessed\_subqueries}(\mathit{filter}_{2,1})$ and $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{2,1})$ (by `transfer`$_5$ via `transfer′`).

4. **$\mathbf{E_6}$**

   Firing the active edge E_6 (by `fire′`$_1$ via `fire′`), the algorithm empties the set $\mathit{unprocessed\_subqueries}_2(\mathit{filter}_{2,1})$ and transfers (x_4, z_4), a fresh variant of (x_3, z), through the edge E_6 and adds nothing to $\mathit{tuples}(\mathit{input\_path})$ (by `transfer`$_2$ via `transfer′`), because there exists $(x_3, y_3) \in \mathit{tuples}(\mathit{input\_path})$, which is more general than any other tuples.

5. **$\mathbf{E_7}$–$\mathbf{E_8}$–$\mathbf{E_9}$–$\mathbf{E_{10}}$**

   Firing the active edge E_7 (by `fire′`), the algorithm transfers (x_3, y_3) through this edge and empties the set $\mathit{unprocessed}(E_7)$. This produces $\{((x_3, y_3), \{x/x_3, y/y_3\})\}$ (by `transfer′`$_3$ via `transfer′`), which is then transferred through the edge E_8, producing $\{((a, b), \varepsilon), ((a, c), \varepsilon), ((c, d), \varepsilon), ((d, a), \varepsilon)\}$ (by `transfer′`$_4$ via `transfer′`), which is then transferred through the edge E_9, producing $\{(a, b), (a, c), (c, d), (d, a)\}$ (by `transfer′`), which in turn is then transferred through the edge E_{10} and added to the empty sets $\mathit{tuples}(\mathit{ans\_path})$, $\mathit{unprocessed}(E_{11})$ and $\mathit{unprocessed}(E_{15})$ (by `transfer`$_1$ via `transfer′`).

6. **$\mathbf{E_{11}}$**

   Firing the active edge E_{11} (by `fire′`), the algorithm transfers $\{(a, b), (a, c), (c, d), (d, a)\}$ through this edge and empties the set $\mathit{unprocessed}(E_{11})$. This adds those tuples to the empty set $\mathit{unprocessed\_tuples}(\mathit{filter}_{2,1})$ (by `transfer′`).

7. $\mathbf{E_{12}}$**–**$\mathbf{E_{13}}$**–**$\mathbf{E_{14}}$

   Firing the active edge E_{12} (by `fire′`$_3$ via `fire′`) and processing the sets *unprocessed_subqueries*$(\mathit{filter}_{2,1})$ and *unprocessed_tuples*$(\mathit{filter}_{2,1})$, the algorithm empties these sets and produces the set of subqueries $\{((a, y_3), \{y/y_3, z/b\}), ((a, y_3), \{y/y_3, z/c\}), ((c, y_3), \{y/y_3, z/d\}), ((d, y_3), \{y/y_3, z/a\})\}$, which is transferred through the edge E_{12}, producing $\{((a, d), \varepsilon), ((c, a), \varepsilon), ((d, b), \varepsilon), ((d, c), \varepsilon)\}$ (by `transfer′`$_4$ via `transfer′`), which is then transferred through the edge E_{13}, producing $\{(a, d), (c, a), (d, b), (d, c)\}$ (by `transfer′`), which in turn is then transferred through the edge E_{14} and added to the sets *tuples*(*ans_path*), *unprocessed*(E_{11}) and *unprocessed*(E_{15}) (by `transfer′`$_1$ via `transfer′`). After these steps, we have:

   – *tuples*(*ans_path*) $= \{(a, b), (a, c), (c, d), (d, a), (a, d), (c, a), (d, b), (d, c)\}$,
   – *unprocessed*$(E_{11}) = \{(a, d), (c, a), (d, b), (d, c)\}$,
   – *unprocessed*(E_{15}) = *tuples*(*ans_path*).

8. $\mathbf{E_{11}}$ **and** $\mathbf{E_{12}}$**–**$\mathbf{E_{13}}$**–**$\mathbf{E_{14}}$

   The algorithm repeatedly fires the edges E_{11} and E_{12} until no new tuple is added to *tuples*(*ans_path*), *unprocessed*(E_{11}) and *unprocessed*(E_{15}). This takes two rounds and after such steps, the edges E_{11} and E_{12} become inactive and we have:

   – *tuples*(*ans_path*) = *unprocessed*$(E_{15}) = \{(a, b), (a, c), (c, d), (d, a), (a, d), (c, a), (d, b), (d, c), (a, a), (c, b), (c, c), (d, d)\}$.

9. $\mathbf{E_{15}}$

   Firing the active edge E_{15} (by `fire′`), the algorithm transfers the aforementioned tuples of *unprocessed*(E_{15}) through this edge and empties the set *unprocessed*(E_{15}). This adds those tuples to the empty set *unprocessed_tuples*$(\mathit{filter}_{3,1})$ (by `transfer′`). After these steps, we have:

   – *subqueries*$(\mathit{filter}_{3,1})$ = *unprocessed_subqueries*$(\mathit{filter}_{3,1}) = \{((x_1, y_1), \{x/x_1, y/y_1\})\}$,
   – *unprocessed_tuples*$(\mathit{filter}_{3,1}) = \{(a, b), (a, c), (c, d), (d, a), (a, d), (c, a), (d, b), (d, c), (a, a), (c, b), (c, c), (d, d)\}$.

10. $\mathbf{E_{16}}$

    Firing the active edge E_{16} (by `fire′`$_3$ via `fire′`) and processing the sets *unprocessed_subqueries*$(\mathit{filter}_{3,1})$ and *unprocessed_tuples*$(\mathit{filter}_{3,1})$, the algorithm empties these sets and produces the set of subqueries $\{((a, b), \{x/a, y/b\}), ((a, c), \{x/a, y/c\}), ((c, d), \{x/c, y/d\}), ((d, a), \{x/d, y/a\}), ((a, d), \{x/a, y/d\}), ((c, a), \{x/c, y/a\}), ((d, b), \{x/d, y/b\}), ((d, c), \{x/d, y/c\}), ((a, a), \{x/a, y/a\}), ((c, b), \{x/c, y/b\}), ((c, c), \{x/c, y/c\}), ((d, d), \{x/d, y/d\})\}$, which is transferred through the edge E_{16} and added to the empty sets *subqueries*$(\mathit{filter}_{3,2})$, *unprocessed_subqueries*$(\mathit{filter}_{3,2})$ and *unprocessed_subqueries*$_2(\mathit{filter}_{3,2})$ (by `transfer`$_5$ via `transfer′`).

11. $\mathbf{E_{17}}$

    Firing the active edge E_{17} (by `fire′`$_1$ via `fire′`), the algorithm empties the set *unprocessed_subqueries*$_2(\mathit{filter}_{3,2})$ and transfers the set of tuples $\{(b, a), (c, a), (d, c), (a, d), (d, a), (a, c), (b, d), (c, d), (a, a), (b, c), (c, c), (d, d)\}$ through the edge E_{17} and adds nothing to *tuples*(*input_path*) (by `transfer`$_2$ via `transfer′`), because there exists $(x_3, y_3) \in$ *tuples*(*input_path*), which is more general than any other tuples.

12. $\mathbf{E_{18}}$–$\mathbf{E_{19}}$

    Observe that the edge E_{18} is active, the current net is stable up to the layer 1 (no edge among E_4–E_{14} is active), and the edge E_{17} is inactive. Thus, selecting the edge E_{18} (for firing) satisfies the admissibility w.r.t. strata's stability. Firing this edge (by `fire'`$_4$ via `fire'`) and processing $\mathit{unprocessed\_subqueries}(\mathit{filter}_{3,2})$, the algorithm empties this set and produces the set of subqueries $\{((a,b),\{\varepsilon\}),\ ((d,b),\{\varepsilon\}),\ ((c,b),\{\varepsilon\})\}$, which is transferred through the edge E_{18}, producing $\{(a,b),\ (d,b),\ (c,b)\}$ (by `transfer'`), which, in turn, is then transferred through the edge E_{19} and added to the empty set $\mathit{tuples}(\mathit{ans\_acyclic})$ (by `transfer`$_1$ via `transfer'`).

At this point, no edge in the net is active. The algorithm terminates and returns the set $\mathit{tuples}(\mathit{ans\_acyclic}) = \{(a,b),\ (d,b),\ (c,b)\}$.

B Data Complexity, Soundness and Completeness

In this section, we prove that the QSQN-STR evaluation method for stratified Datalog$^\neg$ is sound, complete and has a PTIME data complexity.

The following lemma states a property of Algorithm 1.

Lemma 1. *For every intensional predicate r used in P, if $\bar{t} \in \mathit{tuples}(\mathit{ans\_r})$, then $\bar{t}$ is a ground tuple (i.e., a tuple without variables).*

This property follows from the safety conditions of the Datalog$^\neg$ program P. Technically, one can prove it by induction on the moment of adding $\bar{t}$ to $\mathit{tuples}(\mathit{ans\_r})$ and an inner induction on j that, if a subquery $(\bar{t}', \delta)$ is transferred to a node $\mathit{filter}_{i,j}$, where the predicate of A_i is r, then $\mathit{Vars}(\bar{t}') \subseteq \mathit{Vars}((B_{i,j}, \ldots, B_{i,n_i})\delta)$ and, for every $x \in \mathit{Vars}((B_{i,1}, \ldots, B_{i,j-1})) \cap \mathit{Vars}((B_{i,j}, \ldots, B_{i,n_i}))$, $\delta(x)$ is a constant (i.e., δ contains a pair x/c for some constant c). Additionally, as the next step, if a subquery $(\bar{t}, \delta)$ is transferred to the node $\mathit{post\_filter}_i$, then $\delta = \varepsilon$ and $\bar{t}$ is a ground tuple. The proof is straightforward and omitted.

Lemma 2. *Algorithm 1 runs in polynomial time in the size of I.*

Proof. Let n be the size of I. Without loss of generality, we assume that all intensional predicates of the signature are used in P. As the Datalog$^\neg$ program P is fixed, the arities of all intensional predicates are bounded by a constant, and the number of constant symbols occurring in P is also bounded by a constant.

For each intensional predicate p, let $\mathit{all\_tuples}(\mathit{input\_p})$ denote the set of all tuples that are added to $\mathit{tuples}(\mathit{input\_p})$ during the run of the algorithm (including the ones that are deleted from $\mathit{tuples}(\mathit{input\_p})$ at some later steps). The cardinality of $\mathit{all\_tuples}(\mathit{input\_p})$ is bounded by a polynomial in n. The reasons are as follows:

– Let k be the arity of p. Before a tuple $\bar{t}$ is added to $\mathit{tuples}(\mathit{input\_p})$, $\bar{t}$ is not an instance of a fresh variant of any $\bar{t}' \in \mathit{tuples}(\mathit{input\_p})$, hence there exists a renaming substitution $\theta_{\bar{t}}$ such that $\mathit{dom}(\theta_{\bar{t}}) = \mathit{Vars}(\bar{t})$, $\mathit{range}(\theta_{\bar{t}}) \subseteq \{x_1, \ldots, x_k\}$ and $\bar{t}\theta_{\bar{t}}$ is not an instance of a fresh variant of any $\bar{t}' \in \mathit{tuples}(\mathit{input\_p})$. When a tuple is deleted from $\mathit{tuples}(\mathit{input\_p})$, its fresh variant must be an instance of a tuple that will be added to $\mathit{tuples}(\mathit{input\_p})$ at the next step. Hence, if $\{\bar{t}, \bar{t}'\} \subseteq \mathit{all\_tuples}(\mathit{input\_p})$ and $\bar{t} \neq \bar{t}'$, then $\bar{t}\theta_{\bar{t}} \neq \bar{t}'\theta_{\bar{t}'}$.

– The sets $\mathit{all\_tuples}(\mathit{input\_p})$ and $\{\bar{t}\theta_{\bar{t}} \mid \bar{t} \in \mathit{all\_tuples}(\mathit{input\_p})\}$ have the same cardinality, which is bounded by a polynomial in n because each element of the latter set is a k-ary tuple constructed from the variables $x_1, \ldots, x_k$ and the constants occurring in $P \cup I$.

For each intensional predicate p, the cardinality of $\mathit{tuples}(\mathit{ans\_p})$ is also bounded by a polynomial in n. This follows from Lemma 1.

Each "elementary operation" executed by the algorithm is related to a $\bar{t} \in \mathit{all\_tuples}(\mathit{input\_p})$ for some p and can be labeled by the pair $(\bar{t}, p)$, which is chosen so that $\bar{t} \in \mathit{all\_tuples}(\mathit{input\_p})$ is the most direct cause of the operation. Observe that, for each $(\bar{t}, p)$, the number of "elementary operations" executed by the algorithm and labeled by $(\bar{t}, p)$ is bounded by a polynomial in n. As the cardinality of $\mathit{all\_tuples}(\mathit{input\_p})$ for each intensional predicate p is bounded by a polynomial in n, we conclude that the algorithm runs in polynomial time in n. ∎

We will need the well-known Lifting Lemma, whose restriction to Datalog is presented below. Its proof can be found in [9].

Lemma 3 (Lifting Lemma). *Let P be a Datalog program, G a goal and θ a substitution. Suppose there exists an SLD-refutation of $P \cup \{G\theta\}$ using mgu's $\theta_1, \ldots, \theta_n$ such that the variables of the input program clauses are distinct from the variables in G and θ. Then, there exist a substitution γ and an SLD-refutation of $P \cup \{G\}$ using the same sequence of input program clauses, the same selected atoms, and mgu's $\theta'_1, \ldots, \theta'_n$ such that $\theta\theta_1 \ldots \theta_n = \theta'_1 \ldots \theta'_n\gamma$.*

For each predicate p of the signature (now called the *primary signature*), let p' be a new extensional predicate (for playing the role of $\sim p$). For each $1 \leq k \leq K$, let P'_k be the Datalog program obtained from P_k by replacing every $\sim p$ with p'. For each $0 \leq k \leq K$, let $M_k = M_{P_1 \cup \ldots \cup P_k, I}$ and let I_k be the instance of extensional predicates specified as follows:

- if p is an extensional predicate from the primary signature, then $I_k(p) = I(p)$;
- if p is an h-ary predicate from the primary signature, then $I_k(p') = \{\bar{t} \mid \bar{t}$ is an h-ary tuple of constants from $U_{P,I}$ such that $p(\bar{t}) \notin M_k\}$.

Lemma 4. *For every $1 \leq k \leq K$, every intensional predicate p of the primary signature and every tuple $\bar{t}$ of constants, $p(\bar{t}) \in M_k$ iff $p(\bar{t}) \in T_{P'_1 \cup \ldots \cup P'_k, I_{k-1}} \uparrow \omega$.*

This lemma immediately follows from the fact that the standard semantics of stratified Datalog$^\neg$ agrees with the stable model semantics [5].

Lemma 5. *During a run of Algorithm 1, for every intensional predicate r of P with $\mathit{layer}(\mathit{input\_r}) = k$ and for every tuples $\bar{t}$ and $\bar{t}'$ of terms,*

a) if $\bar{t} \in \mathit{tuples}(\mathit{ans\_r})$, then $r(\bar{t}) \in M_{P,I}$,

b) if the QSQ-STR-net is stable up to the layer k, $\bar{t} \in \mathit{tuples}(\mathit{input\_r})$, $r(\bar{t}') \in M_{P,I}$ and $\bar{t}'$ is an instance of $\bar{t}$, then $\bar{t}' \in \mathit{tuples}(\mathit{ans\_r})$.

Proof. We prove this lemma by induction on k. The base case $k = 0$ is trivial.

For the induction step, we first show that a run of the QSQN-STR method for the Datalog$^\neg$ program $P_1 \cup \ldots \cup P_k$ can be treated as a run of the QSQN method for the Datalog program $P'_1 \cup \ldots \cup P'_k$ by considering each $\sim p$ as the extensional predicate p' specified by I_{k-1}. For this, we only need to show that if $\mathit{pred}(A_i) = r$, $\mathit{kind}(\mathit{filter}_{i,j}) = \mathit{intensional}$, $\mathit{neg}(\mathit{filter}_{i,j}) = \mathit{true}$, $\mathit{pred}(\mathit{filter}_{i,j}) = p$, $\mathit{layer}(\mathit{input\_p}) = h$ and $h < k$, then:

(i) if $\bar{t} \in \mathit{tuples}(\mathit{ans\_p})$, then $p'(\bar{t}) \notin I_{k-1}$,

(ii) if $(\bar{t}_{j-1}, \delta_{j-1}) \in \mathit{subqueries}(\mathit{filter}_{i,j})$, then for every tuple $\bar{t}$ of constants from $U_{P,I}$ such that $p(\bar{t})$ is an instance of $\mathit{atom}(\mathit{filter}_{i,j})\delta_{j-1}$ and $p'(\bar{t}) \notin I_{k-1}$, $\bar{t}$ was added by Algorithm 1 to $\mathit{tuples}(\mathit{ans\_p})$ at some step before the subquery $(\bar{t}_{j-1}, \delta_{j-1})$ is processed for the edge $(\mathit{filter}_{i,j}, \mathit{succ}(\mathit{filter}_{i,j}))$.

Assume that the premises of the main implication hold. Consider the assertion (i) and assume that $\bar{t} \in \mathit{tuples}(\mathit{ans\_p})$. By the inductive assumption (a), $p(\bar{t}) \in M_{P,I}$. Thus, $p(\bar{t}) \in M_{k-1}$ and hence $p(\bar{t}) \notin I_{k-1}$. For the assertion (ii), assume that $(\bar{t}_{j-1}, \delta_{j-1}) \in \mathit{subqueries}(\mathit{filter}_{i,j})$, $\bar{t}$ is a tuple of constants from $U_{P,I}$ such that $p(\bar{t})$ is an instance of $\mathit{atom}(\mathit{filter}_{i,j})\delta_{j-1}$ and $p'(\bar{t}) \notin I_{k-1}$. We have that $p(\bar{t}) \in M_{k-1}$, and hence $p(\bar{t}) \in M_{P,I}$. Since the used control strategy is admissible w.r.t. strata's stability, before calling $\mathtt{fire}'(\mathit{filter}_{i,j}, \mathit{succ}(\mathit{filter}_{i,j}))$, the subquery $(\bar{t}_{j-1}, \delta_{j-1})$ has already been processed for the edge $(\mathit{filter}_{i,j}, \mathit{input\_p})$ and, as a consequence, $\mathit{tuples}(\mathit{input\_p})$ contains a tuple $\bar{t}''$ such that a fresh variant of $\mathit{atom}(\mathit{filter}_{i,j})\delta_{j-1}$ is an instance of $p(\bar{t}'')$. Thus, $\bar{t}$ is an instance of $\bar{t}''$. Furthermore, at that moment the QSQ-STR-net is stable up to the layer h. By the inductive assumption (b) for h instead of k and p, $\bar{t}''$, $\bar{t}$ instead of r, $\bar{t}$, $\bar{t}'$, respectively, we have that $\bar{t} \in \mathit{tuples}(\mathit{ans\_p})$, which completes the proof of the assertion (ii).

By the assertions (i) and (ii), we can now treat a run of Algorithm 1 on the part consisting of the layers up to k of the QSQ-STR-net as a run of the QSQN method on a QSQ-net by considering each $\sim p$ as the extensional predicate p' specified by I_{k-1}.[1]

Consider the assertion (a) and assume that the premise of the implication holds. Since $\bar{t} \in \mathit{tuples}(\mathit{ans\_r})$, by the soundness of the QSQN method for Datalog (see [1, Lemma 4.2] for the case when $l = 0$ and $T(p) = \mathit{false}$ for every intensional predicate p), $\bar{t}$ is a correct answer for $(P'_1 \cup \ldots \cup P'_k) \cup I_{k-1} \cup \{\leftarrow r(\bar{t})\}$. By the correctness of the fixpoint semantics of positive logic program (see, e.g., [6, Theorems 6.5 and 6.6]), it follows that $r(\bar{t}) \in T_{P'_1 \cup \ldots \cup P'_k, I_{k-1}} \uparrow \omega$. Hence, by Lemma 4, $r(\bar{t}) \in M_k$, which implies $r(\bar{t}) \in M_{P,I}$. This completes the proof of the assertion (a).

Consider the assertion (b) and assume that the premises of the implication hold. Since $r(\bar{t}') \in M_{P,I}$, we have that $r(\bar{t}') \in M_k$, and by Lemma 4, $r(\bar{t}') \in T_{P'_1 \cup \ldots \cup P'_k, I_{k-1}} \uparrow \omega$. By the correctness of the fixpoint semantics of positive logic program (see, e.g., [6, Theorems 6.5 and 6.6]) and the completeness of SLD-resolution (see, e.g., [6, Theorems 8.6]), there exists an SLD-refutation of $(P'_1 \cup \ldots \cup P'_k) \cup I_{k-1} \cup \{\leftarrow r(\bar{t}')\}$ with mgu's $\theta_1, \ldots, \theta_n$. Since $\bar{t}'$ is an instance of $\bar{t}$, there exists a substitution θ such that $\bar{t}' = \bar{t}\theta$. By the Lifting Lemma 3, there exists an SLD-refutation of $(P'_1 \cup \ldots \cup P'_k) \cup I_{k-1} \cup \{\leftarrow r(\bar{t})\}$ with mgu's $\theta'_1, \ldots, \theta'_n$ such that $\theta\theta_1 \ldots \theta_n = \theta'_1 \ldots \theta'_n\delta$ for some substitution δ. By the completeness of the QSQN method for Datalog (see [1, Lemma 4.3] for the case when $l = 0$ and $T(p) = \mathit{false}$ for every intensional predicate p), $\bar{t}\theta'_1 \ldots \theta'_n$ is an instance of a fresh variant of some tuple $\bar{t}'' \in \mathit{tuples}(\mathit{ans\_r})$. Since $\bar{t}' = \bar{t}'\theta_1 \ldots \theta_n = \bar{t}\theta\theta_1 \ldots \theta_n = \bar{t}\theta'_1 \ldots \theta'_n\delta$ is an instance of $\bar{t}\theta'_1 \ldots \theta'_n$, $\bar{t}'$ is also an instance of $\bar{t}''$. Since $\bar{t}''$ is a ground tuple (by Lemma 1), it follows that $\bar{t}' = \bar{t}''$. This completes the proof of the assertion (b). ■

Corollary 1. *After a run of Algorithm 1 for a query $q(\bar{x})$ to a stratified Datalog$^\neg$ database (P, I), for every tuple of terms $\bar{t}$, $\bar{t} \in \mathit{tuples}(\mathit{ans\_q})$ iff $q(\bar{t}) \in M_{P,I}$.*

This corollary immediately follows from Lemma 5. Together with Lemma 2, it implies the following theorem.

Theorem 1. *The QSQN-STR method formulated by Algorithm 1 for evaluating queries to stratified Datalog$^\neg$ databases is sound, complete and has a PTIME data complexity.*

C Preliminary Experiments

We have implemented a prototype of QSQN-STR in Java, using a control strategy named IDFS2, which is specified in [1]. We have made a comparison between our prototype of QSQN-STR and Datalog Educational System (DES – a deductive database system) [7] w.r.t. the number of generated tuples in the answer relations that correspond to intensional predicates. The experimental results given in [1, Section 6.5] show that the number of generated tuples in the *answer* relations that correspond to negated intensional predicates in the case of QSQN-STR is often smaller than the one in the case of DES.

Our prototype of QSQN-STR [2] has not yet been optimized. So, in general, it cannot compete with highly optimized engines like XSB [8]. Nevertheless, we have performed experiments and

[1] The edge $(\mathit{filter}_{i,j}, \mathit{input\_p})$ may cause adding more tuples to $\mathit{tuples}(\mathit{input\_p})$, but they do not affect the soundness and completeness of the QSQN method ([1, Lemmas 4.2 and 4.3] for the case when $l = 0$ and $T(p) = \mathit{false}$ for every intensional predicate p).

made a comparison between our prototype of QSQN-STR, DES-DBMS[2] (version 5.0.1) and SWI-Prolog[3] (version 6.4) w.r.t. the execution time by using a number of tests. Comparing our prototype of QSQN-STR with other existing engines is time-consuming and left as future work.[4]

In this section, we present the mentioned comparison between our prototype of QSQN-STR, DES-DBMS and SWI-Prolog. In general, such a comparison does not reveal much about advantages of the used evaluation methods, because these systems use different programming styles and/or languages. Besides, QSQN-STR is a framework that allows every control strategy admissible w.r.t. strata's stability, and our prototype of QSQN-STR adopts the control strategy IDFS2 [1], which may be further improved. The point is not the efficiency of our prototype of QSQN-STR. The aim of our experiments and comparison is only to support the claim that QSQN-STR is a useful evaluation framework for stratified Datalog$^\neg$.

Our prototype of QSQN-STR uses extensional relations stored in a MySQL database. DES-DBMS was also implemented in Java using SQL DBMS'. SWI-Prolog is a well-known logic programming software, which can easily be connected to a MySQL database through an ODBC driver. Without using a MySQL database for storing extensional relations, SWI-Prolog runs very fast. For a fair comparison, however, we performed tests with SWI-Prolog using extensional relations stored in the same MySQL database as for QSQN-STR. For the tests with QSQN-STR, we set $T(v) = \mathit{false}$ for each $v = \mathit{filter}_{i,j} \in V$ with $\mathit{pred}(v) = \mathit{extensional}$.

Let P_1 be the stratified Datalog$^\neg$ program consisting of the following clauses, where link_1, link_2, origin and $\mathit{destination}$ are extensional predicates, $\mathit{reachable}_1$, $\mathit{reachable}_2$, $\mathit{reachable}$, query_1 and query_2 are intensional predicates, x, y and z are variables:

$$\mathit{reachable}_1(x,y) \leftarrow \mathit{link}_1(x,y), \tag{1}$$
$$\mathit{reachable}_1(x,y) \leftarrow \mathit{link}_1(x,z), \mathit{reachable}_1(z,y), \tag{2}$$

$$\mathit{reachable}_2(x,y) \leftarrow \mathit{link}_2(x,y), \tag{3}$$
$$\mathit{reachable}_2(x,y) \leftarrow \mathit{link}_2(x,z), \mathit{reachable}_2(z,y), \tag{4}$$

$$\mathit{reachable}(x,y) \leftarrow \mathit{reachable}_1(x,y), \tag{5}$$
$$\mathit{reachable}(x,y) \leftarrow \mathit{reachable}_2(x,y), \tag{6}$$

$$\mathit{query}_1(x,y) \leftarrow \mathit{origin}(x), \mathit{destination}(y), {\sim}\mathit{reachable}(x,y), \tag{7}$$
$$\mathit{query}_2(x,y) \leftarrow \mathit{origin}(x), \mathit{destination}(y), \mathit{reachable}(x,y), {\sim}\mathit{reachable}(y,x). \tag{8}$$

Let P_2 be the stratified Datalog$^\neg$ program that differs from P_1 in that the clauses (2) and (4) are replaced by the following one, with i being 1 or 2, respectively:

$$\mathit{reachable}_i(x,y) \leftarrow \mathit{reachable}_i(x,z), \mathit{link}_i(z,y).$$

Similarly, let P_3 be the stratified Datalog$^\neg$ program that differs from P_1 in that the clauses (2) and (4) are replaced by the following one, with i being 1 or 2, respectively:

$$\mathit{reachable}_i(x,y) \leftarrow \mathit{reachable}_i(x,z), \mathit{reachable}_i(z,y).$$

[2] The Datalog Education System (DES) with a DBMS via ODBC, available at `http://des.sourceforge.net` (see also, e.g., [7]).

[3] Available at `http://www.swi-prolog.org/`

[4] XSB is known as an efficient engine for in-memory Datalog$^\neg$ databases due to the suspension-resumption mechanism, advantages of WAM (Warren Abstract Machine) and other optimizations. We think that our prototype of QSQN-STR cannot compete with XSB when the computation can totally be done in the memory without accessing to the secondary storage. For a comparison with XSB, at least we want to run XSB using very large extensional relations stored on disk. However, at the moment we have a technical problem with connecting XSB to a MySQL DBMS via an ODBC driver. We could not find time for comparing our prototype of QSQN-STR with other engines like DLV [29], NP Datalog [24] and *clasp* [19]. In general, we think that those systems were designed and implemented to deal, among others, with answer set programming (ASP) and, as the main aim of ASP engines is to find an answer set (i.e., a stable model) for a given logic program, they are not goal-driven and, in general, not as efficient as expected for answering queries to stratified Datalog$^\neg$ databases. Of course, without performing experiments and comparisons, nothing can be formally stated.

Let I_1 be the extensional instance specified as follows, where n is a parameter and elements like $a_{i,j}$, o_k and d_k are constant symbols:

$$\begin{aligned}
I_1(\mathit{origin}) &= \{o_k \mid 1 \leq k \leq n\},\\
I_1(\mathit{destination}) &= \{d_k \mid 1 \leq k \leq n\},\\
I_1(\mathit{link}_1) &= \{(o_k, a_{1,1}), (a_{i,1}, a_{i+1,1}), (a_{n,1}, d_k) \mid 1 \leq k \leq n, 1 \leq i < n\},\\
I_1(\mathit{link}_2) &= \{(o_k, a_{1,j}), (a_{i,j}, a_{i+1,j}), (a_{n,j}, d_k) \mid 1 \leq k \leq n, 1 \leq i < n, 1 \leq j \leq n\}.
\end{aligned}$$

Let I_2 be the extensional instance specified as follows:

$$\begin{aligned}
I_2(\mathit{origin}) &= I_1(\mathit{origin}),\\
I_2(\mathit{destination}) &= I_1(\mathit{destination}),\\
I_2(\mathit{link}_1) &= I_1(\mathit{link}_1) \cup \{(a_{i+1,1}, a_{i,1}) \mid 1 \leq i < n\},\\
I_2(\mathit{link}_2) &= I_1(\mathit{link}_2) \cup \{(a_{i+1,j}, a_{i,j}) \mid 1 \leq i < n, 1 \leq j \leq n\}.
\end{aligned}$$

The extensional instances I_1 and I_2 are illustrated in Figures 2 and 3, respectively. We consider the tests specified as follows.

Test	**Program**	**Extensional Instance**
Test 1	P_1	I_1
Test 2	P_1	I_2
Test 3	P_2	I_1
Test 4	P_2	I_2
Test 5	P_3	I_1
Test 6	P_3	I_2

For each of the tests, we consider the following values of n:

$$1)\ n = 20, \quad 2)\ n = 40, \quad 3)\ n = 60, \quad 4)\ n = 80, \quad 5)\ n = 100$$

and the following queries (cf. [3, Remark 2.3]):

$$a)\ \mathit{query}_1(x, y), \qquad b)\ \mathit{query}_1(o_1, d_1), \qquad c)\ \mathit{query}_2(x, y), \qquad d)\ \mathit{query}_2(o_1, d_1).$$

Our experiments were done using a computer with Windows 10 (64-bit), Intel® Core$^{\text{TM}}$ i5-6500 CPU 3.20 GHz and 8 GB RAM. Figures 4–9 show a comparison between our prototype of QSQN-STR, SWI-Prolog and DES-DBMS w.r.t. the execution time for the mentioned tests. The experimental results of SWI-Prolog are shown only for the tests for which SWI-Prolog can terminate properly.[5] For each of the mentioned engines, each test was executed 10 times and the average value of execution time in milliseconds was taken. To give a better data visualization, the execution times are shown after being converted by $\log_{10}$. Detailed instructions for verifying our experiments are included in [2].

The results presented in Figures 4–9 show that our prototype of QSQN-STR outperforms DES-DBMS by a few orders of magnitude in term of execution time for all of the tests. It is competitive with SWI-Prolog for the tests for which SWI-Prolog can terminate properly.

[5] SWI-Prolog uses SLDNF-resolution, which can have infinite derivations even for Datalog, and for such cases SWI-Prolog terminates with the communication “out of local stack”.

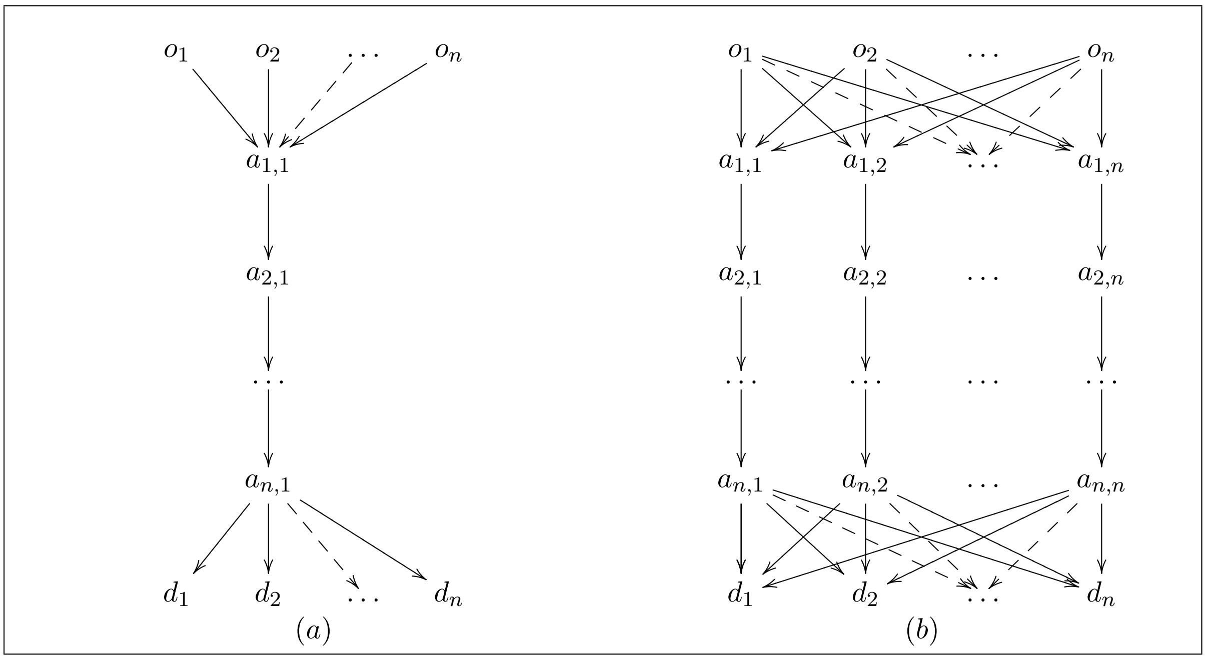

o_1 o_2 $\dots$ o_n
$a_{1,1}$
$a_{2,1}$
$\dots$
$a_{n,1}$
d_1 d_2 $\dots$ d_n
(a)
o_1 o_2 $\dots$ o_n
$a_{1,1}$ $a_{1,2}$ $\dots$ $a_{1,n}$
$a_{2,1}$ $a_{2,2}$ $\dots$ $a_{2,n}$
$\dots$ $\dots$ $\dots$ $\dots$
$a_{n,1}$ $a_{n,2}$ $\dots$ $a_{n,n}$
d_1 d_2 $\dots$ d_n
(b)

Fig. 2. The extensional instance I_1: (a) $I_1(link_1)$, and (b) $I_1(link_2)$

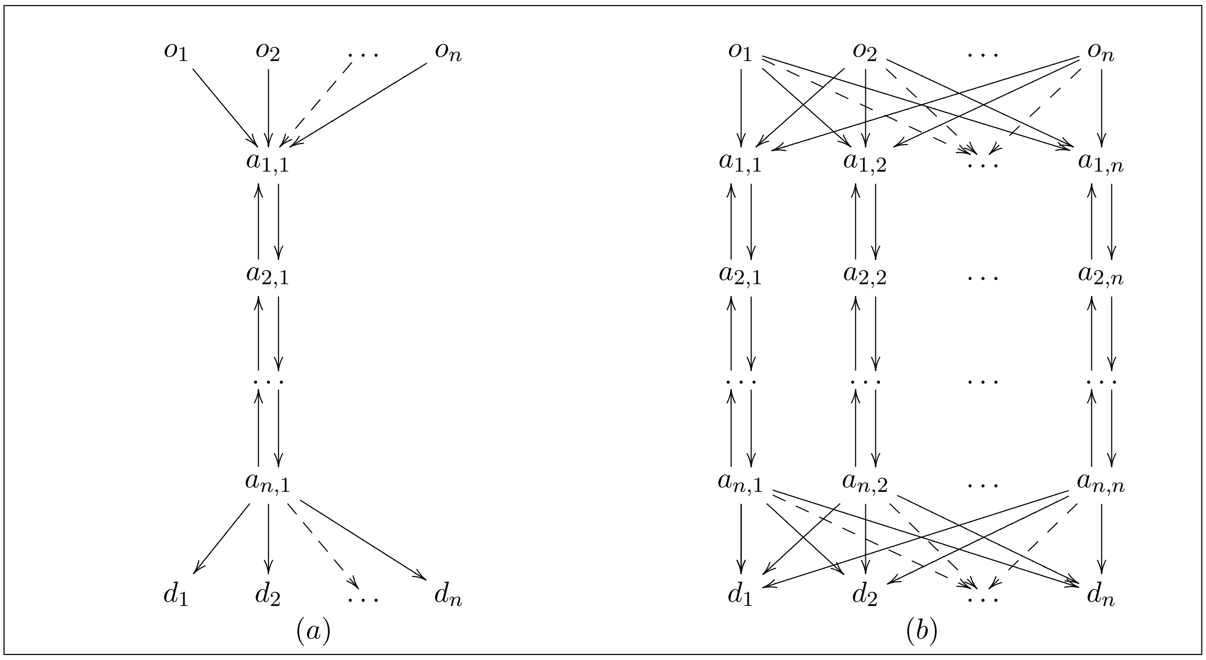

o_1 o_2 $\dots$ o_n
$a_{1,1}$
$a_{2,1}$
$\dots$
$a_{n,1}$
d_1 d_2 $\dots$ d_n
(a)
o_1 o_2 $\dots$ o_n
$a_{1,1}$ $a_{1,2}$ $\dots$ $a_{1,n}$
$a_{2,1}$ $a_{2,2}$ $\dots$ $a_{2,n}$
$\dots$ $\dots$ $\dots$ $\dots$
$a_{n,1}$ $a_{n,2}$ $\dots$ $a_{n,n}$
d_1 d_2 $\dots$ d_n
(b)

Fig. 3. The extensional instance I_2: (a) $I_2(link_1)$, and (b) $I_2(link_2)$

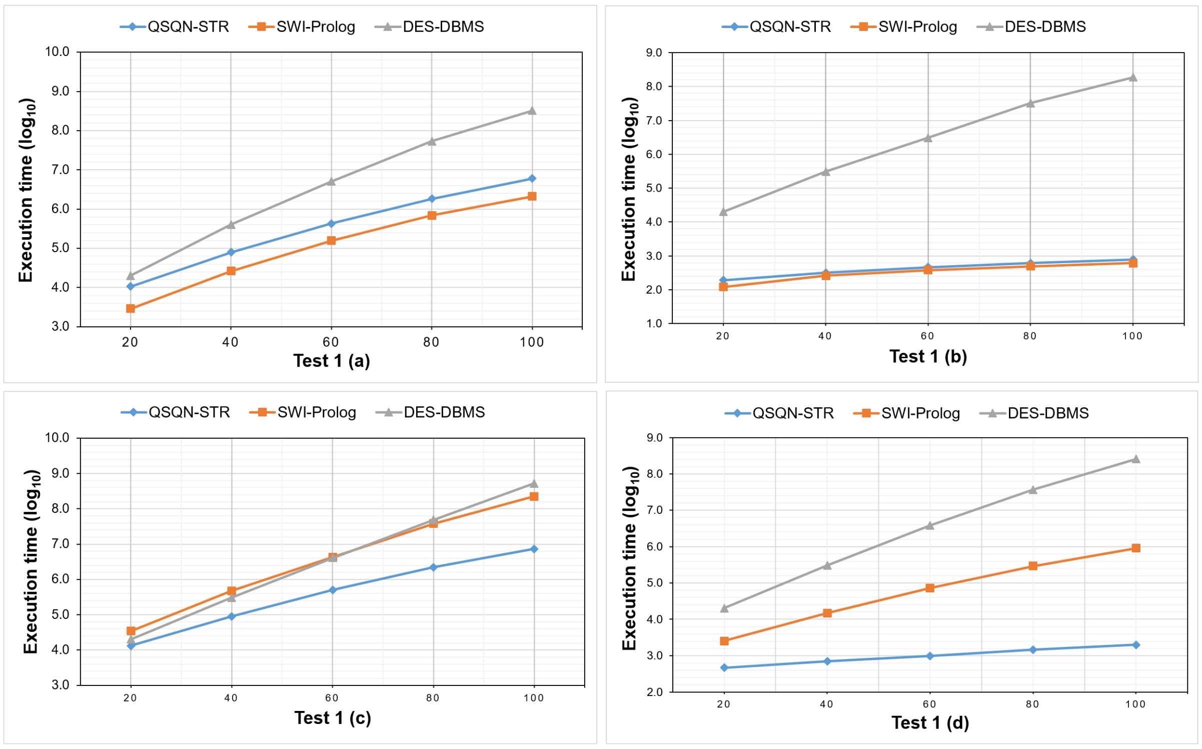


Fig. 4. Experimental results for Test 1

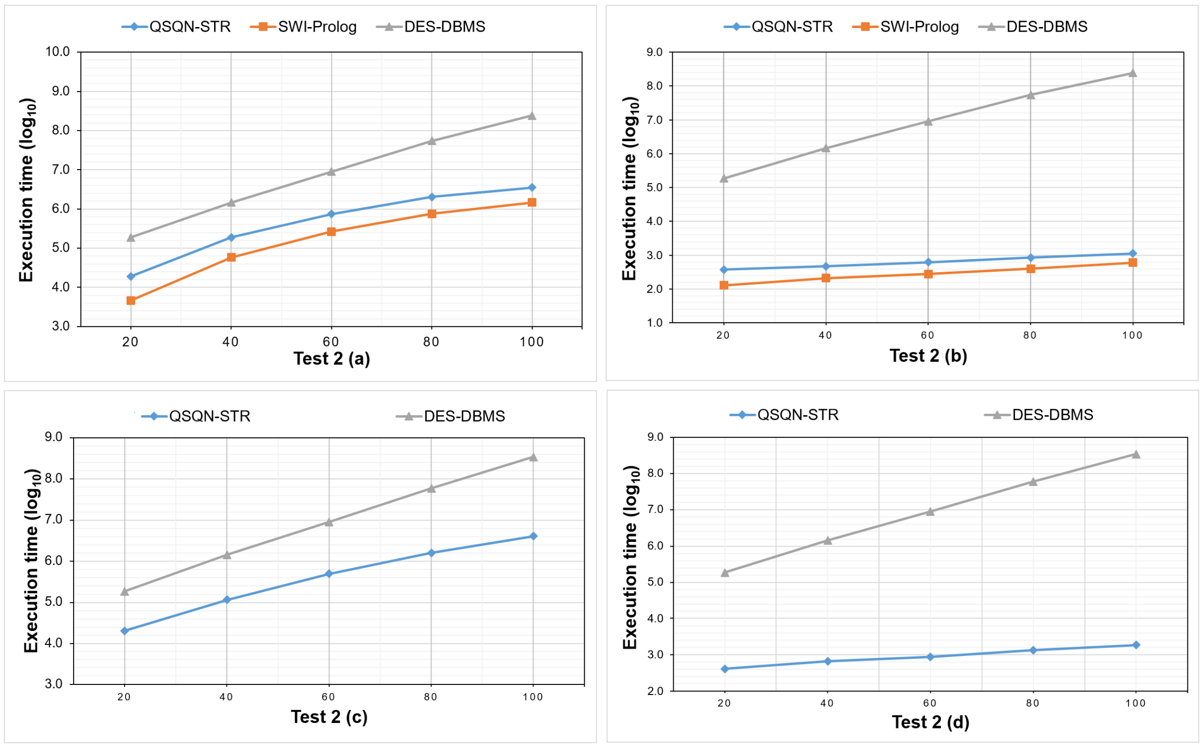


Fig. 5. Experimental results for Test 2

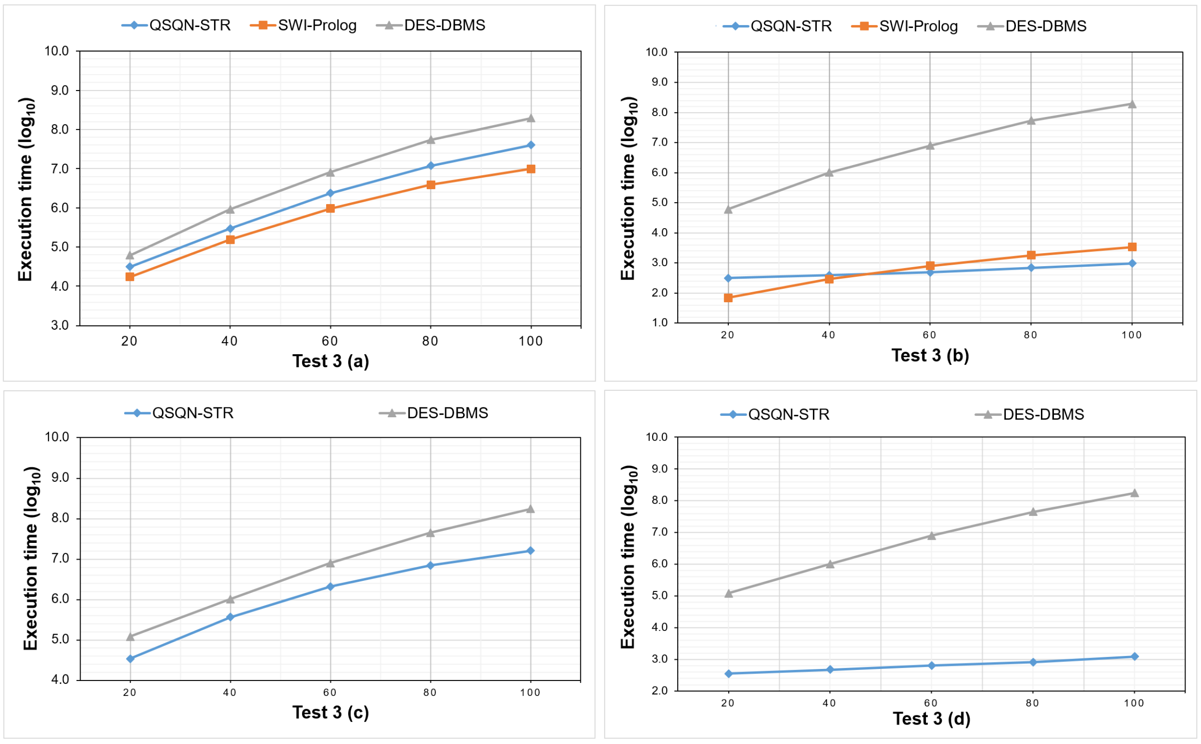


Fig. 6. Experimental results for Test 3

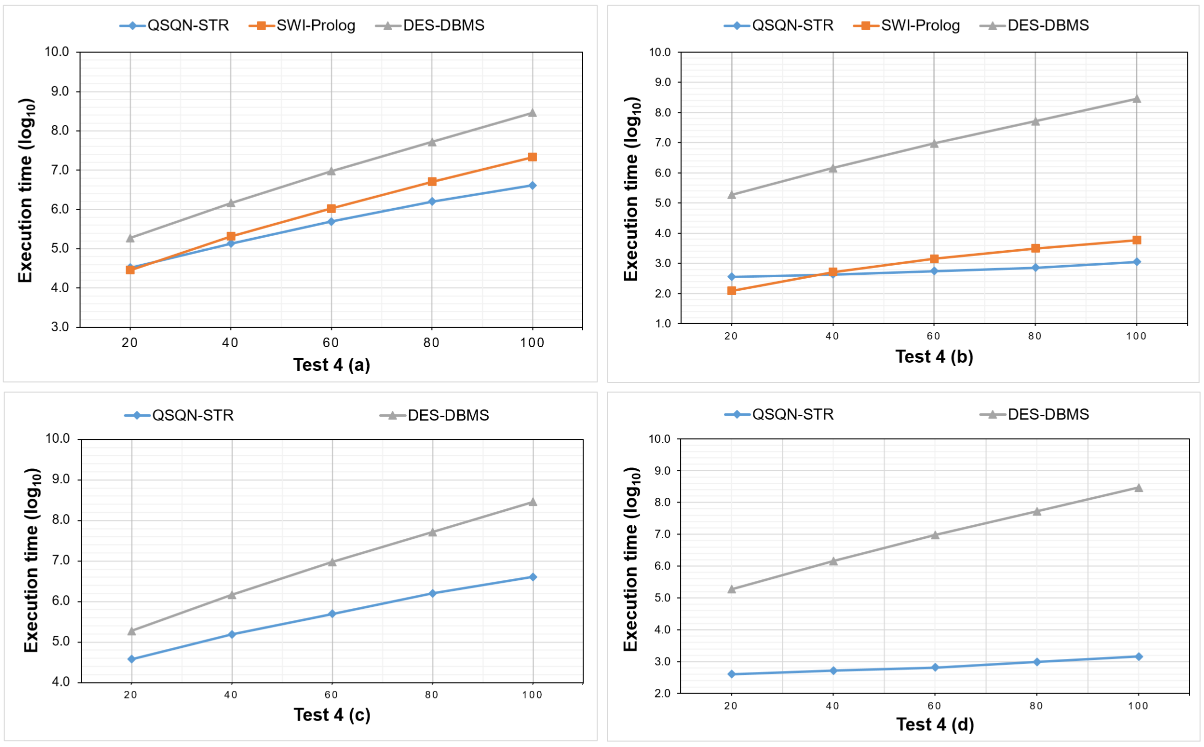


Fig. 7. Experimental results for Test 4

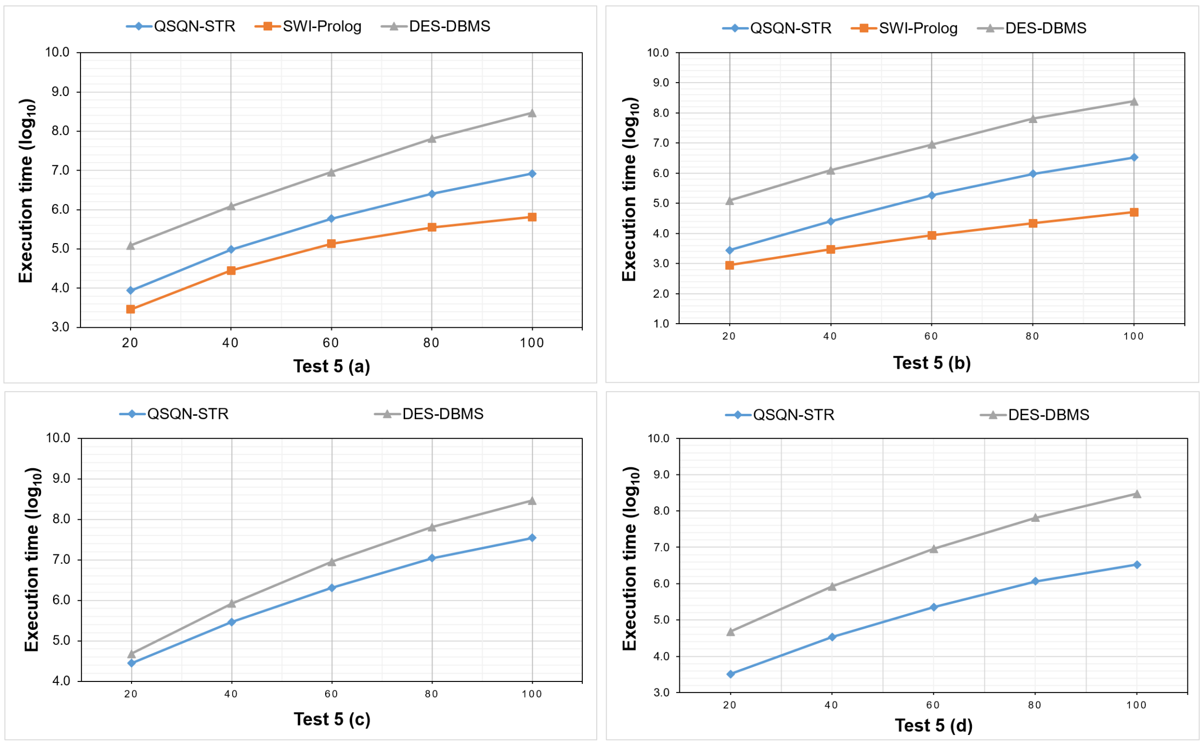


Fig. 8. Experimental results for Test 5

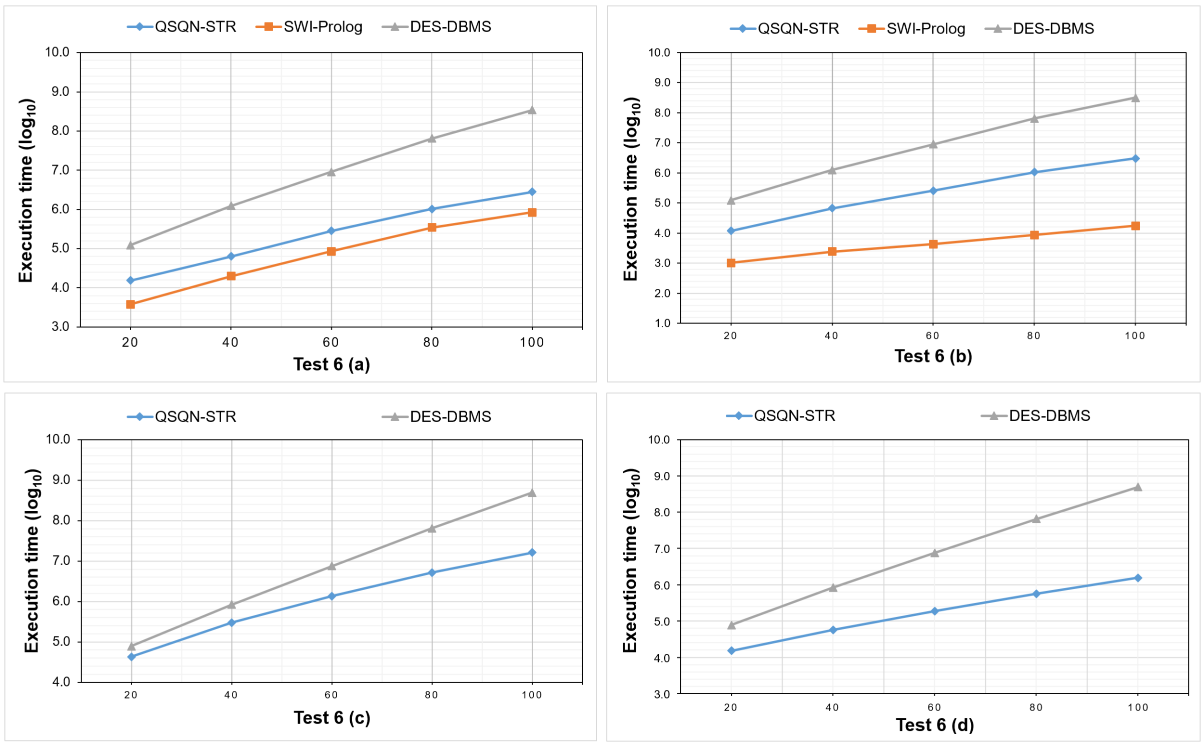


Fig. 9. Experimental results for Test 6

References

1. Cao, S. T.: Methods for Evaluating Queries to Horn Knowledge Bases in First-Order Logic. Ph.D. thesis, University of Warsaw, 2016.
2. Cao, S. T.: An Implementation in Java of the QSQN-STR Evaluation Method. Available at: http://mimuw.edu.pl/~sonct/QSQN-STR18.zip, 2018.
3. Cao, S. T.—Nguyen, L. A.: Incorporating Stratified Negation into Query-Subquery Nets for Evaluating Queries to Stratified Deductive Databases. To appear in Computing and Informatics, 2018, doi: 10.1007/978-3-319-17996-4_32.
4. Cao, S. T.—Nguyen, L. A.: Online Appendix: A Demonstration of the QSQN-STR Method for evaluating Queries to Stratified Datalog$^\neg$. Available at: http://mimuw.edu.pl/~nguyen/QSQN-STR-demonstration.pdf, 2018.
5. Gelfond, M.—Lifschitz, V.: The Stable Model Semantics for Logic Programming. Proceedings of ICLP/SLP 1988, pp. 1070–1080, MIT Press, 1988.
6. Lloyd, J. W.: Foundations of Logic Programming. 2nd edition, Springer, 1987, doi: 10.1007/978-3-642-83189-8.
7. Sáenz-Pérez, F.: DES: A Deductive Database System. Electronic Notes in Theoretical Computer Science, Vol. 271, 2011, pp. 63–78, doi: 10.1016/j.entcs.2011.02.011.
8. Sagonas, K. F.—Swift, T.—Warren, D. S.: XSB as an Efficient Deductive Database Engine. Proceedings of SIGMOD Conference 1994, pp. 442–453, ACM Press, 1994.
9. Staab, S.: Completeness of the SLD-Resolution. Slides of a Course on Advanced Data Modeling. Available at: https://west.uni-koblenz.de/files/SS08/adm08/db2-ss08-slides9.ppt, 2008.

D Pseudocode of the QSQN-STR Method for Stratified Datalog$^\neg$

Algorithm 1: evaluating a query $q(\overline{x})$ to a stratified Datalog$^\neg$ database (P, I).

Input: a stratified Datalog$^\neg$ database (P, I), a stratification $P = P_1 \cup \ldots \cup P_K$ of P and a query $q(\overline{x})$.

Output: the set of all correct answers for the query $q(\overline{x})$ on (P, I).

```
1 let $(V, E, T)$ be a QSQN-STR structure of $P$;
  // $T$ can be chosen arbitrarily or appropriately
2 set $C$ so that $(V, E, T, C)$ is an empty QSQ-STR-net of $P$;
3 let $\overline{x}'$ be a fresh variant of $\overline{x}$;
4 $tuples(input\_q) := \{\overline{x}'\}$;
5 foreach $(input\_q, v) \in E$ do $unprocessed(input\_q, v) := \{\overline{x}'\}$;
6 while there exists $(u, v) \in E$ such that active-edge$(u, v)$ holds do
7     select any edge $(u, v) \in E$ such that active-edge$(u, v)$ holds and the
      selection satisfies the admissibility w.r.t. strata's stability;
8     fire$'(u, v)$;
9 return $tuples(ans\_q)$;
```

```
Function active-edge($(u, v)$)
  Global data: a QSQ-STR-net $(V, E, T, C)$ of $P$.
  Input: an edge $(u, v) \in E$.
  Output: $true$ if there are data to transfer through the edge $(u, v)$, and $false$ otherwise.
1 if $u$ is $pre\_filter_i$ or $post\_filter_i$ then return $false$;
2 else if $u$ is $input\_p$ or $ans\_p$ then return $unprocessed(u, v) \neq \emptyset$;
3 else if $u$ is $filter_{i,j}$ and $kind(u) = extensional$ then
4     return $T(u) = true \wedge unprocessed\_subqueries(u) \neq \emptyset$;
5 else // $u$ is of the form $filter_{i,j}$ and $kind(u) = intensional$
6     let $p = pred(u)$;
7     if $v = input\_p$ then return $unprocessed\_subqueries_2(u) \neq \emptyset$;
8     else return $unprocessed\_subqueries(u) \neq \emptyset \vee unprocessed\_tuples(u) \neq \emptyset$;
```

```
Procedure add-subquery($\bar{t}, \delta, \Gamma, v$)
  Purpose: add the subquery $(\bar{t}, \delta)$ to $\Gamma$, but keep in $\Gamma$ only the most general subqueries
           w.r.t. $v$.
1 if no subquery in $\Gamma$ is more general than $(\bar{t}, \delta)$ w.r.t. $v$ then
2     delete from $\Gamma$ all subqueries less general than $(\bar{t}, \delta)$ w.r.t. $v$;
3     add $(\bar{t}, \delta)$ to $\Gamma$;
```

```
Procedure add-tuple($\bar{t}, \Gamma$)
  Purpose: add a fresh variant of the tuple $\bar{t}$ to $\Gamma$, but keep in $\Gamma$ only the most general
           tuples.
1 let $\bar{t}'$ be a fresh variant of $\bar{t}$;
2 if $\bar{t}'$ is not an instance of any tuple from $\Gamma$ then
3     delete from $\Gamma$ all tuples that are instances of $\bar{t}'$;
4     add $\bar{t}'$ to $\Gamma$;
```

```
Procedure transfer'($D, u, v$)
  Global data: a stratified Datalog$^\neg$ database $(P, I)$ and a QSQ-STR-net $(V, E, T, C)$ of
               $P$.
  Input: data $D$ to transfer through the edge $(u, v) \in E$.
1 if $D = \emptyset$ then return;
2 else if $v$ is $post\_filter_i$ then transfer'($\{\bar{t} \mid (\bar{t}, \varepsilon) \in D\}, v, succ(v)$);
3 else if $u$ is $ans\_p$ then $unprocessed\_tuples(v) := unprocessed\_tuples(v) \cup D$;
4 else if $v$ is $ans\_p$ then transfer$_1$($D, u, v$);
5 else if $v$ is $input\_p$ then transfer$_2$($D, u, v$);
6 else if $u$ is $input\_p$ then transfer'$_3$($D, u, v$);
7 else if $v$ is $filter_{i,j}$, $kind(v) = extensional$ and $T(v) = false$ then
8     if $neg(v) = false$ then transfer'$_4$($D, u, v$);
9     else transfer'$_{4b}$($D, u, v$);
10 else transfer$_5$($D, u, v$);
```

Procedure transfer$_1(D, u, v)$

```
// $v = ans\_p$, $u = post\_filter_i$, $D$ is a set of tuples of constants
1 foreach $\bar{t} \in D - tuples(v)$ do
2     add $\bar{t}$ to $tuples(v)$;
3     foreach $(v, w) \in E$ do add $\bar{t}$ to $unprocessed(v, w)$;
```

Procedure transfer$_2(D, u, v)$

```
// $v = input\_p$, $u = filter_{i,j}$ and $kind(u) = intensional$
1 foreach $\bar{t} \in D$ do
2     let $\bar{t}'$ be a fresh variant of $\bar{t}$;
3     if $\bar{t}'$ is not an instance of any tuple from $tuples(v)$ then
4         delete from $tuples(v)$ all tuples that are instances of $\bar{t}'$;
5         add $\bar{t}'$ to $tuples(v)$;
6         foreach $(v, w) \in E$ do
7             delete from $unprocessed(v, w)$ all tuples that are instances of $\bar{t}'$;
8             add $\bar{t}'$ to $unprocessed(v, w)$;
```

Procedure transfer$'_3(D, u, v)$

```
// $u$ is $input\_p$ and $v = pre\_filter_i$
1 $\Gamma := \emptyset$;
2 foreach $\bar{t} \in D$ do
3     if $p(\bar{t})$ and $atom(v)$ are unifiable by an mgu $\gamma$ then
4         add-subquery($\bar{t}\gamma, \gamma_{|post\_vars(v)}, \Gamma, succ(v)$);
5 transfer'($\Gamma, v, succ(v)$);
```

Procedure transfer$'_4(D, u, v)$

```
// $v = filter_{i,j}$, $kind(v) = extensional$, $T(v) = \mathtt{false}$ and $neg(v) = \mathtt{false}$
1 let $p = pred(v)$ and set $\Gamma := \emptyset$;
2 foreach $(\bar{t}, \delta) \in D$ and $\bar{t}' \in I(p)$ do
3     if $atom(v)\delta$ and $p(\bar{t}')$ are unifiable by an mgu $\gamma$ then
4         add-subquery($\bar{t}\gamma, (\delta\gamma)_{|post\_vars(v)}, \Gamma, succ(v)$);
5 transfer'($\Gamma, v, succ(v)$);
```

Procedure transfer$'_{4b}(D, u, v)$

```
// $v = filter_{i,j}$, $kind(v) = extensional$, $T(v) = \mathtt{false}$, $neg(v) = \mathtt{true}$
1 let $p = pred(v)$ and set $\Gamma := \emptyset$;
2 foreach $(\bar{t}, \delta) \in D$ do
3     if $atom(v)\delta \notin \{p(\bar{t}') \mid \bar{t}' \in I(p)\}$ then
4         add-subquery($\bar{t}, \delta_{|post\_vars(u)}, \Gamma, succ(v)$);
5 transfer'($\Gamma, v, succ(v)$);
```

Procedure transfer$_5(D, u, v)$

```
/* v = filter_{i,j} , (kind(v) = extensional and T(v) = true or kind(v) = intensional)
   and u is not of the form ans_p                                                   */
1 foreach $(\bar{t}, \delta) \in D$ do
2     if no subquery in subqueries$(v)$ is more general than $(\bar{t}, \delta)$ w.r.t. $v$ then
3         delete from subqueries$(v)$ and unprocessed_subqueries$(v)$ all subqueries less
            general than $(\bar{t}, \delta)$ w.r.t. $v$;
4         add $(\bar{t}, \delta)$ to both subqueries$(v)$ and unprocessed_subqueries$(v)$;
5         if kind$(v)$ = intensional then
6             delete from unprocessed_subqueries$_2(v)$ all subqueries less general than $(\bar{t}, \delta)$
                w.r.t. $v$;
7             add $(\bar{t}, \delta)$ to unprocessed_subqueries$_2(v)$;
```

Procedure fire$'(u, v)$

```
Global data: a stratified Datalog$^\neg$ database $(P, I)$ and a QSQ-STR-net $(V, E, T, C)$ of
             $P$.
Input: an edge $(u, v) \in E$ such that active-edge$(u, v)$ holds.
1  if $u$ is ans_p then
2      transfer'(unprocessed$(u, v), u, v)$;
3      unprocessed$(u, v) := \emptyset$;
4  else if $u$ is input_p then
5      transfer'(unprocessed$(u, v)$ − tuples(ans_p)$, u, v)$;
6      unprocessed$(u, v) := \emptyset$;
7  else if $v$ is input_p then fire'$_1(u, v)$;
8  else if $u$ is filter$_{i,j}$ and neg$(u)$ = false then
9      if kind$(u)$ = extensional then fire'$_2(u, v)$;
10     else fire'$_3(u, v)$;
11 else if $u$ is filter$_{i,j}$ and neg$(u)$ = true then fire'$_4(u, v)$;
```

Procedure fire$'_1(u, v)$

```
// v = input_p, u = filter_{i,j} and kind(u) = intensional
1 let $p$ = pred$(u)$ and set $\Gamma := \emptyset$;
2 foreach $(\bar{t}, \delta) \in$ unprocessed_subqueries$_2(u)$ do
3     let $p(\bar{t}') = atom(u)\delta$;
4     add-tuple$(\bar{t}', \Gamma)$;
5 unprocessed_subqueries$_2(u) := \emptyset$;
6 transfer'$(\Gamma, u, v)$;
```

Procedure fire$'_2(u, v)$

```
// $u = filter_{i,j}$, $neg(u) = \texttt{false}$, $kind(u) = extensional$ and $T(u) = \texttt{true}$
1 let $p = pred(u)$ and set $\Gamma := \emptyset$;
2 foreach $(\bar{t}, \delta) \in unprocessed\_subqueries(u)$ and $\bar{t}' \in I(p)$ do
3     if $atom(u)\delta$ and $p(\bar{t}')$ are unifiable by an mgu $\gamma$ then
4         add-subquery$(\bar{t}\gamma, (\delta\gamma)_{|post\_vars(u)}, \Gamma, v)$;
5 $unprocessed\_subqueries(u) := \emptyset$;
6 transfer$'(\Gamma, u, v)$;
```

Procedure fire$'_3(u, v)$

```
// $u = filter_{i,j}$, $neg(u) = \texttt{false}$, $kind = intensional$ and $v = succ(u)$
1 let $p = pred(u)$ and set $\Gamma := \emptyset$;
2 foreach $(\bar{t}, \delta) \in unprocessed\_subqueries(u)$ and $\bar{t}' \in tuples(ans\_p)$ do
3     if $atom(u)\delta$ and $p(\bar{t}')$ are unifiable by an mgu $\gamma$ then
4         add-subquery$(\bar{t}\gamma, (\delta\gamma)_{|post\_vars(u)}, \Gamma, v)$;
5 $unprocessed\_subqueries(u) := \emptyset$;
6 foreach $(\bar{t}, \delta) \in subqueries(u)$ and $\bar{t}' \in unprocessed\_tuples(u)$ do
7     if $atom(u)\delta$ and $p(\bar{t}')$ are unifiable by an mgu $\gamma$ then
8         add-subquery$(\bar{t}\gamma, (\delta\gamma)_{|post\_vars(u)}, \Gamma, v)$;
9 $unprocessed\_tuples(u) := \emptyset$;
10 transfer$'(\Gamma, u, v)$;
```

Procedure fire$'_4(u, v)$

```
// $u = filter_{i,j}$, $neg(u) = \texttt{true}$ and $v = succ(u)$
1 let $p = pred(u)$ and set $\Gamma := \emptyset$;
2 let $R$ be $I(p)$ if $kind(u) = extensional$, and $tuples(ans\_p)$ otherwise;
3 foreach $(\bar{t}, \delta) \in unprocessed\_subqueries(u)$ do
4     if $atom(u)\delta \notin \{p(\bar{t}') \mid \bar{t}' \in R\}$ then
5         add-subquery$(\bar{t}, \delta_{|post\_vars(u)}, \Gamma, v)$;
6 $unprocessed\_subqueries(u) := \emptyset$;
7 transfer$'(\Gamma, u, v)$;
```