Parallel-Correctness and Containment for Conjunctive Queries with Union and Negation

Gaetano Geck¹, Bas Ketsman∗², Frank Neven³, and Thomas Schwentick⁴

1 TU Dortmund University, Dortmund, Germany
2 Hasselt University, Hasselt, Belgium; and Transnational University of Limburg, Belgium/The Netherlands
3 Hasselt University, Hasselt, Belgium; and Transnational University of Limburg, Belgium/The Netherlands
4 TU Dortmund University, Dortmund, Germany

Abstract

Single-round multiway join algorithms first reshuffle data over many servers and then evaluate the query at hand in a parallel and communication-free way. A key question is whether a given distribution policy for the reshuffle is adequate for computing a given query, also referred to as parallel-correctness. This paper extends the study of the complexity of parallel-correctness and its constituents, parallel-soundness and parallel-completeness, to unions of conjunctive queries with and without negation. As a by-product it is shown that the containment problem for conjunctive queries with negation is coNEXPTIME-complete.

1998 ACM Subject Classification H.2.3 Query Languages, H.2.4 Distributed databases

Keywords and phrases Conjunctive queries, distributed evaluation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.9

1 Introduction

Motivated by recent in-memory systems like Spark [7] and Shark [21], Koutris and Suciu introduced the massively parallel communication model (MPC) [15] where computation proceeds in a sequence of parallel steps each followed by global synchronisation of all servers. Of particular interest in the MPC model are queries that can be evaluated in one round of communication [9]. In its most naïve setting, a query \( Q \) is evaluated by reshuffling the data over many servers, according to some distribution policy, and then computing \( Q \) at each server in a parallel but communication-free manner. A notable family of distribution policies is formed within the Hypercube algorithm [3, 9, 11]. A property of Hypercube distributions is that for any instance \( I \), the central execution of \( Q(I) \) always equals the union of the evaluations of \( Q \) at every computing node (or server). The latter guarantees the correctness of the distributed evaluation for any conjunctive query by the Hypercube algorithm.

Ameloot et al. [4] introduced a general framework for reasoning about one-round evaluation algorithms under arbitrary distribution policies. They introduced parallel-correctness as a property of a query w.r.t. a distribution policy which states that central execution always equals distributed execution, that is, equals the union of the evaluations of the query at each server under the given distribution policy. One of the main results of [4] is that deciding

---

² PhD Fellow of the Research Foundation – Flanders (FWO).

© Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick;
licensed under Creative Commons License CC-BY
19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume; Article No. 9: pp. 9:1–9:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
Parallel-correctness for conjunctive queries (CQs) is $\Pi^P_2$-complete under arbitrary distribution policies. The upper bound follows rather directly from a semantical characterisation of parallel-correctness in terms of properties of minimal valuations. Specifically, it was shown that a conjunctive query is parallel-correct w.r.t. a distribution policy, if the distribution policy sends for every minimal valuation its required facts to at least one node.

As union and negation are fundamental operators, we extend in this paper the study of parallel-correctness to unions of conjunctive queries (UCQ), conjunctive queries with negation (CQ$^-$), and unions of conjunctive queries with negation (UCQ$^-$). In fact, we study two additional but related notions: parallel-soundness and parallel-completeness. While parallel-correctness implies equivalence between centralised and distributed execution, parallel-soundness (respectively, parallel-completeness) requires that distributed execution is contained in (respectively, contains) centralised execution. Of course, parallel-soundness and parallel-completeness together are equivalent to parallel-correctness. Furthermore, since all monotone queries are parallel-sound, on this class parallel-correctness is equivalent to parallel-completeness.

We start by investigating parallel-correctness for UCQ. Interestingly, for a UCQ to be parallel-correct under a certain distribution policy it is not required that every disjunct is parallel-correct. We extend the characterisation for parallel-correctness in terms of minimal valuations for CQs to UCQs and thereby obtain membership in $\Pi^P_2$. The matching lower bound follows, of course, from the lower bound for CQs [4].

Next, we study parallel-correctness for (unions of) conjunctive queries with negation. Sadly, when negation comes into play, parallel-correctness can no longer be characterised in terms of properties of valuations. Instead our algorithms are based on counter-examples of exponential size, yielding coNEXPTIME upper bounds. It turns out that this is optimal, though, as our corresponding lower bounds show. The proof of the lower bounds comes along an unexpected route: we exhibit a reduction from query containment for CQ$^-$ to parallel-correctness of CQ$^-$ (and its two variants) and show that query containment for CQ$^-$ is coNEXPTIME-complete. This is considerably different from what we thought was folklore knowledge of the community. Indeed, the $\Pi^P_2$-completeness result for query containment for CQ$^-$ mentioned in [19] only seems to hold for fixed database schemas (or a fixed arity bound, for that matter). We note that Mugnier et al. [17] provide a $\Pi^P_2$ upper bound proof for CQ$^-$ containment and explicitly mention that it holds under the assumption that the arity of predicates is bounded by a constant. Altogether, parallel-correctness (and its variants) for (unions of) conjunctive queries with negation is thus complete for coNEXPTIME.

Finally, a natural question is how the high complexity of parallel-correctness in the presence of negation can be lowered. We identify two cases in which the complexity drops. More specifically, the complexity decreases from coNEXPTIME to $\Pi^P_2$ if the database schema is fixed or the arity of relations is bounded, and to coNP for unions of full conjunctive queries with negation. In the latter case, we again employ a reduction from containment of full conjunctive queries (with negation) and obtain novel results on the containment problem in this setting as well. All upper bounds hold for queries with inequalities.

Outline. This paper is further organised as follows. In Section 2, we discuss related work. In Section 3, we introduce the necessary definitions. We address parallel-correctness for unions of conjunctive queries in Section 4. We consider containment of conjunctive queries with negation in Section 5 and parallel-correctness together with its variants in Section 6. We discuss the restriction to full conjunctive queries in Section 7. We conclude in Section 8. Missing proof details can be found in the full version of this paper [14].
2 Related work

As mentioned in the introduction, Koutris and Suciu introduced the massively parallel communication model (MPC) [15]. A key property is that computation proceeds in a sequence of parallel steps, each followed by global synchronisation of all computing nodes. In this model, evaluation of conjunctive queries [8, 15] and skyline queries [2] has been considered. Beame, Koutris and Suciu [9] proved a matching upper and lower bound for the amount of communication needed to compute a full conjunctive query without self-joins in one communication round. The upper bound is provided by a randomised algorithm called Hypercube which uses a technique that can be traced back to Ganguly, Silberschatz, and Tsur [13] and is described in the context of map-reduce by Afrati and Ullman [3].

Ameloot et al. [4] introduced a general framework for reasoning about one-round evaluation algorithms under arbitrary distribution policies. They introduced the notion of parallel-correctness and proved its associated decision problem to be \(\Pi_p^2\)-complete for conjunctive queries. In addition, towards optimisation in MPC, they considered parallel-correctness transfer. Here, parallel-correctness transfers from \(Q\) to \(Q'\) when \(Q'\) is parallel-correct under every distribution policy for which \(Q\) is parallel-correct. The associated decision problem for conjunctive queries is shown to be \(\Pi_p^2\)-complete. In addition, some restricted cases (e.g., transferability under Hypercube distributions), are shown to be NP-complete.

Our definition of a distribution policy is borrowed from Ameloot et al. [5] (but already surfaces in the work of Zinn et al. [22]), where distribution policies are used to define the class of policy-aware transducer networks. The work by Ameloot et al. [6, 5] relates coordination-free computation with definability in variants of Datalog. One-round communication algorithms in MPC can be seen as very restrictive coordination-free computation.

The complexity of query containment for conjunctive queries is proved to be NP-complete by Chandra and Merlin [10]. Levy and Sagiv provide a test for query containment of conjunctive queries with negation [16] that involves exploring an exponential number of possible counter-example instances. In the context of information integration, Ullman [19] gives a comprehensive overview of query containment (with and without negation) and states the complexity of query containment for CQ\(^-\) to be \(\Pi_p^2\)-complete. As mentioned in the introduction, the latter apparently only holds when the database schema is fixed or the arity of relations is considered to be bounded. A proof for the \(\Pi_p^2\)-lowerbound is given by Farré et al. [12]. Based on [16], Wei and Lausen [20] study a method for testing containment that exploits containment mappings for the positive parts of queries, and additionally provide a characterisation for UCQ\(^-\) containment.

3 Definitions

3.1 Queries and instances

We assume an infinite set \(\text{dom}\) of data values that can be represented by strings over some fixed alphabet. By \(\text{dom}_n\) we denote the set of data values represented by strings of length at most \(n\). A database schema \(\mathcal{D}\) is a finite set of relation names \(R\), each with some arity \(ar(R)\). We also write \(R^{(k)}\) as a shorthand to denote that \(R\) is a relation of arity \(k\). We call \(R(t)\) a fact when \(R\) is a relation name and \(t\) a tuple over \(\text{dom}\) of appropriate arity. We say that a fact \(R(t)\) is over a database schema \(\mathcal{D}\) if \(R \in \mathcal{D}\). For a subset \(U \subseteq \text{dom}\) we write \(\text{facts}(\mathcal{D}, U)\) for the set of possible facts over schema \(\mathcal{D}\) and \(U\) and by \(\text{facts}(\mathcal{D})\) we denote \(\text{facts}(\mathcal{D}, \text{dom})\). A (database) instance \(I\) over \(\mathcal{D}\) is a finite set of facts over \(\mathcal{D}\). By \(\text{adom}(I)\) we denote the set of data values occurring in \(I\). A query \(Q\) over input schema \(\mathcal{D}_1\) and output
schema $D_2$ is a generic mapping from instances over $D_1$ to instances over $D_2$. Genericity means that for every permutation $\pi$ of $\text{dom}$ and every instance $I$, $Q(\pi(I)) = \pi(Q(I))$. We say that $Q$ is contained in $Q'$, denoted $Q \subseteq Q'$, iff for all instances $I$, $Q(I) \subseteq Q'(I)$.

### 3.2 Unions of conjunctive queries with negation

Let $\text{var}$ be an infinite set of variables, disjoint from $\text{dom}$. An atom over schema $D$ is of the form $R(x)$, where $R$ is a relation name from $D$ and $x = (x_1, \ldots, x_k)$ is a tuple of variables in $\text{var}$ with $k = \text{ar}(R)$. A conjunctive query $Q$ with negation and inequalities over input schema $D$ is an expression of the form

$$T(x) \leftarrow R_1(y_1), \ldots, R_m(y_m), \neg S_1(z_1), \ldots, \neg S_n(z_n), \beta_1, \ldots, \beta_p$$

where all $R_i(y_i)$ and $S_j(z_j)$ are atoms over $D$, every $\beta_i$ is an inequality of the form $s \neq s'$ where $s, s'$ are distinct variables occurring in some $y_i$ or $z_j$, and $T(x)$ is an atom for which $T \notin D$. Additionally, for safety, we require that every variable in $x$ occurs in some $y_i$ and that every variable occurring in a negated atom has to occur in a positive atom as well (safe negation). We refer to the head atom $T(x)$ as head$Q$, to the set \{ $R_1(y_1), \ldots, R_m(y_m), S_1(z_1), \ldots, S_n(z_n)$ \} as body$Q$, and to the set \{ $\beta_1, \ldots, \beta_p$ \} as ineq$Q$. Specifically, we refer to \{ $R_1(y_1), \ldots, R_m(y_m)$ \} as the positive atoms in $Q$, denoted pos$Q$, and to \{ $S_1(z_1), \ldots, S_n(z_n)$ \} as the negated atoms of $Q$, denoted neg$Q$. We denote by vars($Q$) the set of all variables occurring in $Q$. We refer to the class of conjunctive queries with negation and inequalities by $\text{CQ} \text{-- \#}$, its restriction to queries without inequalities, and without negated atoms, and without both by $\text{CQ} \text{-- }\neg$, $\text{CQ} \text{\#}$, and $\text{CQ}$, respectively. As a shorthand we refer to queries from $\text{CQ} \text{-- \#}$ as $\text{CQ} \text{-- \#}_s$ and similarly for the other classes.

A pre-valuation for a $\text{CQ} \text{-- \#} Q$ is a total function $V: \text{vars}(Q) \rightarrow \text{dom}$, which naturally extends to atoms and sets of atoms. It is consistent for $Q$, if $V(\text{pos}_Q) \cap V(\text{neg}_Q) = \emptyset$, and $V(s) \neq V(s')$, for every inequality $s \neq s'$ of $Q$, in which case it is called a valuation. Of course, for a conjunctive query without negated atoms and without inequalities, every pre-valuation is also a valuation. We refer to $V(\text{pos}_Q)$ as the facts required by $V$, and to $V(\text{neg}_Q)$ as the facts prohibited by $V$.

A valuation $V$ satisfies $Q$ on instance $I$ if all facts required by $V$ are in $I$ while no fact prohibited by $V$ is in $I$, that is, if $V(\text{pos}_Q) \subseteq I$ and $V(\text{neg}_Q) \cap I = \emptyset$. In that case, $V$ derives the fact $V(\text{head}_Q)$. The result of $Q$ on instance $I$, denoted $Q(I)$, is defined as the set of facts that can be derived by satisfying valuations for $Q$ on $I$.

A union of conjunctive queries with negation and inequalities is a finite union of $\text{CQ} \text{-- \#}_s$. That is, $Q$ is of the form $\bigcup_{i=1}^{n} Q_i$ where all subqueries $Q_1, \ldots, Q_n$ have the same relation name in their head atoms. We assume disjoint variable sets among different disjuncts in $Q$. That is, $\text{vars}(Q_i) \cap \text{vars}(Q_j) = \emptyset$ for $i \neq j$ and, in particular, $\text{vars}(\text{head}_Q) \neq \text{vars}(\text{head}_Q_i)$. By varmax($Q$) we denote the maximum number of variables that occurs in any disjunct of $Q$. By $\text{UCQ} \text{-- \#}$ we denote the class of unions of conjunctive queries with negation and inequalities and its fragments are denoted correspondingly.

A $\text{CQ} \text{-- \#}$ is called full if all of its variables occur in its head. A $\text{UCQ} \text{-- \#}$ is full if all its subqueries are full.

The result of $Q$ on instance $I$ is $Q(I) = \bigcup_{i=1}^{n} Q_i(I)$. Accordingly, a mapping from variables to data values is a valuation for a $\text{UCQ} \text{-- \#} Q$ if it is a valuation for one of its subqueries.
3.3 Networks, data distribution, and policies

A network \( \mathcal{N} \) is a nonempty finite set of values from \( \text{dom} \), which we call (computing) nodes (or servers). A distribution policy \( P = (U, \text{facts}_P) \) for a database schema \( \mathcal{D} \) and a network \( \mathcal{N} \) consists of a universe \( U \) and a total function \( \text{facts}_P \) that maps each node of \( \mathcal{N} \) to a set of facts from \( \text{facts}(\mathcal{D}, U) \). A node \( \kappa \) is responsible for fact \( f \) (under policy \( P \)) if \( f \in \text{facts}_P(\kappa) \). As a shorthand (and slight abuse of notation), we denote the set of nodes \( \kappa \) that are responsible for some given fact \( f \) by \( P(f) \). For a distribution policy \( P \) and an instance \( I \) over \( \mathcal{D} \), let \( \text{loc-inst}_{P, I} \) denote the function that maps each \( \kappa \in \mathcal{N} \) to \( I \cap \text{facts}_P(\kappa) \), that is, the set of facts in \( I \) for which \( \kappa \) is responsible. We sometimes refer to a given instance \( I \) as the global instance and to \( \text{loc-inst}_{P, I}(\kappa) \) as the local instance at node \( \kappa \).

We note that for some facts from \( \text{facts}(\mathcal{D}, U) \) there are no responsible nodes. This gives our framework some additional flexibility. However, it does not affect our results: in the lower bound proofs we only use distributions for which all facts from \( \text{facts}(\mathcal{D}, U) \) have some responsible nodes. Each distribution policy implicitly induces a network and each query implicitly defines a database (sub-) schema. Therefore, we often omit the explicit notation for networks and schemas.

Given some policy \( P \) that is defined over a network \( \mathcal{N} \), the result \( [Q, P](I) \) of the distributed evaluation of a query \( Q \) on an instance \( I \) in one round is defined as the union of the results of the query evaluated on each node’s local instance. Formally,

\[
[Q, P](I) \overset{\text{def}}{=} \bigcup_{\kappa \in \mathcal{N}} Q(\text{loc-inst}_{P, I}(\kappa)).
\]

In the decision problem for parallel correctness (to be formalised later), the input consists of a query \( Q \) and a distribution policy \( P \). However, it is not obvious how distribution policies should be specified. In principle, they could be defined in an arbitrary fashion, but it is reasonable to assume that given a potential fact \( f \), a node \( \kappa \) and a policy \( P \), it is not too hard to find out whether \( \kappa \) is responsible for \( f \) under \( P \).

For UCQ\( ^k \)'s, which are monotone, our complexity results are remarkably robust with respect to the choice of the representation of distribution policies. In fact, the complexity results coincide for the two extreme possible choices that we consider in this article. In the first case, distribution policies are specified by an explicit list of tuple-node-pairs, whereas in the second case the test whether a given node is responsible for a given tuple can be carried out by a non-deterministic polynomial-time algorithm. However, we do require that some bound \( n \) on the length of strings that represent node names and data values is given. Without such a restriction, no upper complexity bounds would be possible as nodes with names of super-polynomial length in the size of the input would not be accessible.

Considering queries with negated atoms, however, these two settings (seem to) differ, complexity-wise. The reason is that testing parallel-correctness in this setting requires counter examples of size exponential in the size of the query which can not be succinctly represented by policies in \( \mathcal{P}_{\text{fin}} \). We therefore introduce the class \( \mathcal{P}_{\text{rule}} \) allowing for a more economic rule based description of policies. In particular, in \( \mathcal{P}_{\text{rule}} \), the universe \( U \) of a policy is explicitly enumerated and the responsibilities are defined by simple constraints (described below). The latter representation enjoys the same complexity properties as the full NP-test based case.

Now we give more precise definitions of classes of policies and their representations as inputs of algorithmic problems. As said before, policies \( P = (U, \text{facts}_P) \) from \( \mathcal{P}_{\text{fin}} \) are specified by an explicit enumeration of \( U \) and of all pairs \( (\kappa, f) \) where \( \kappa \in P(f) \). A policy \( P = (U, \text{facts}_P) \) from \( \mathcal{P}_{\text{rule}} \) is given by an explicit enumeration of \( U \) and a list of \( \text{rules} \) of the form \( \rho = (A, \kappa) \), where \( A \) is an atom with variables and/or constants from \( U \),
and a network node $\kappa$. The semantics of such a rule is as follows: for every substitution $\mu : \text{var} \cup \text{dom} \to \text{dom}$ that maps variables to values from $U$ and leaves constants from $U$ unchanged, the node $\kappa$ is responsible for the fact $\mu(A)$. A rule is a fact rule if its atom does not contain any variables, that is, $A = R(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n \in U$. In particular, $\mathcal{P}_\text{fin} \subseteq \mathcal{P}_\text{rule}$.

\begin{itemize}
\item \textbf{Example 1.} Let distribution policy $P$ over schema $\{\text{Rel}^{(3)}\}$ and network $\{\kappa_1, \kappa_2\}$ be given by $U = \{1, \ldots, 10\}$ and the rules $\{\text{Rel}(1, x, x), \kappa_1\}$. On global instance $I = \{\text{Rel}(1, 7, 7), \text{Rel}(1, 7, 8), \text{Rel}(2, 9, 8), \text{Rel}(2, 9, 9)\}$, policy $P$ induces local instances $\text{loc-inst}_{P, f}(\kappa_1) = \{\text{Rel}(1, 7, 7)\}$ and $\text{loc-inst}_{P, f}(\kappa_2) = \{\text{Rel}(2, 9, 8), \text{Rel}(2, 9, 9)\}$.
\end{itemize}

The most general classes of policies allow to specify policies by means of a ‘test algorithm’ with time bound $\ell^k$, where $\ell$ is the length of the input and $k$ some constant. Such an algorithm decides, for an input consisting of a node $\kappa$ and fact $f$, whether $\kappa$ is responsible for $f$.\footnote{We note that it is important that for each class of policies there is a fixed $k$ that bounds the exponent in the test algorithm as otherwise we could not expect a polynomial bound for all policies of that class.} A policy $P = (U, r\text{facts}_P)$ from $\mathcal{P}_{\text{npoly}}$ is specified by a pair $(n, A_P)$, where $n$ is a natural number in unary representation and $A_P$ is a non-deterministic algorithm.\footnote{For concreteness, say, a non-deterministic Turing machine.} The universe $U$ of $P$ is the set of all data values that can be represented by strings of length at most $n$ (for some given fixed alphabet) and the underlying network consists of all nodes which are represented by strings of length at most $n$, that is, $N = \text{dom}_n$. A node $\kappa$ is responsible for a fact $f$ if $A_P$, on input $(\kappa, f)$, has an accepting run of at most $|f|^k$ steps. Clearly, each policy of $\mathcal{P}_\text{fin}$ can be described in $\mathcal{P}_{\text{npoly}}^\mathcal{D}$. Let $\mathcal{P}_{\text{npoly}}$ denote the set $\{\mathcal{P}_{\text{npoly}}^\mathcal{D} | \mathcal{D} \subseteq N\}$ of distribution policies and by $\mathcal{P}$ the set $\{\mathcal{P}_\text{fin}, \mathcal{P}_\text{rule}\}$.

\subsection{3.4 Parallel-correctness, soundness, and completeness}

In this paper, we mainly consider the one-round evaluation algorithm for a query $Q$ that first distributes (reshuffles) the data over the computing nodes according to $P$, then evaluates $Q$ in a parallel step at every computing node, and finally outputs all facts that are obtained in this way.\footnote{Since ‘linear time’ is a subtle notion, we rather not consider $\mathcal{P}_{\text{npoly}}^1$.} As formalised next, the one-round evaluation algorithm is correct (sound, complete) if the query $Q$ is parallel-correct (parallel-sound, parallel-complete) under $P$.

\begin{itemize}
\item \textbf{Definition 2.} Let $Q$ be a query, $I$ an instance, and $P$ a distribution policy.
\begin{itemize}
\item $Q$ is parallel-sound on $I$ under $P$ if $Q(I) \supseteq [Q, P]|(I)$.
\item $Q$ is parallel-complete on $I$ under $P$ if $Q(I) \subseteq [Q, P]|(I)$; and,
\item $Q$ is parallel-correct on $I$ under $P$ if $Q(I) = [Q, P]|(I)$, that is, if it is parallel-sound and parallel-complete.
\end{itemize}
\end{itemize}
\begin{itemize}
\item \textbf{Definition 3.} A query $Q$ is parallel-correct (respectively, parallel-sound and parallel-complete) under distribution policy $P = (U, r\text{facts}_P)$, if $Q$ is parallel-correct (respectively, parallel-sound and parallel-complete) on all instances $I \subseteq \text{facts}(\mathcal{D}, U)$.
\end{itemize}

In [4], parallel-correctness is characterised in terms of minimal valuations as defined next:

\begin{itemize}
\item \textbf{Definition 4.} Let $Q$ be a CQ. A valuation $V$ for $Q$ is minimal for $Q$ if there exists no valuation $V'$ for $Q$ such that $V(\text{head}_Q) = V'(\text{head}_Q)$ and $V'(\text{body}_Q) \subseteq V(\text{body}_Q)$.
\end{itemize}
The following lemma is key in obtaining the $\Pi^2_2$ upper bound on the complexity of testing parallel-correctness for conjunctive queries:

**Lemma 5** (Characterisation of parallel-correctness for CQs [4]). A CQ $Q$ is parallel-correct under distribution policy $P = (U, rfacts_P)$ if and only if the following holds:

For every minimal valuation $V$ for $Q$ over $U$, there is a node $\kappa \in \mathcal{N}$ such that $V(\text{body}_Q) \subseteq rfacts_P(\kappa)$.  \hfill (C1)

**Remark 6.** Informally, condition (C1) states that there is a node in the network where all facts required for $V$ meet.

### 3.5 Algorithmic problems

We consider the following decision problems for various sub-classes $C$ and $C'$ of $\text{UCQ}^{\neg,\neq}$ and classes $\mathcal{P}$ of distribution policies from $\{\mathcal{P}_{\text{fin}}, \mathcal{P}_{\text{rule}}\} \cup \mathcal{P}_{\text{poly}}$.

<table>
<thead>
<tr>
<th><strong>Containment</strong>$(C, C')$:</th>
<th><strong>Parallel-Sound</strong>$(C, \mathcal{P})$:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> $Q \in C$ and $Q' \in C'$</td>
<td><strong>Input:</strong> $Q \in C$, $P \in \mathcal{P}$</td>
</tr>
<tr>
<td><strong>Question:</strong> Is $Q \subseteq Q'$?</td>
<td><strong>Question:</strong> Is $Q$ parallel-sound under $P$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Parallel-Complete</strong>$(C, \mathcal{P})$:</th>
<th><strong>Parallel-Correct</strong>$(C, \mathcal{P})$:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> $Q \in C$, $P \in \mathcal{P}$</td>
<td><strong>Input:</strong> $Q \in C$, $P \in \mathcal{P}$</td>
</tr>
<tr>
<td><strong>Question:</strong> Is $Q$ parallel-complete under $P$?</td>
<td><strong>Question:</strong> Is $Q$ parallel-correct under $P$?</td>
</tr>
</tbody>
</table>

### 4 Parallel-correctness: unions of conjunctive queries

Parallel-correctness of unions of conjunctive queries (without negation) reduces to parallel-completeness for the simple reason that these queries are monotone and therefore parallel-sound for every distribution policy. We show below that parallel-completeness remains in $\Pi^2_2$. Hardness already follows from $\Pi^2_2$-hardness of $\text{Parallel-Correct}(CQ, \mathcal{P}_{\text{fin}})$ [4].

As a UCU is parallel-complete under a policy $P$ when all its disjuncts are, it might be tempting to assume that this condition is also necessary. However, as the following example illustrates, this is not the case.

**Example 7.** Let $Q = Q_1 \cup Q_2$, where $Q_1$ and $Q_2$ are the following CQs:

\[
\begin{align*}
Q_1 : & \quad H(x, x) \leftarrow R(x, x), \\
Q_2 : & \quad H(y, z) \leftarrow R(y, z), S(y, z).
\end{align*}
\]

Further, let $P$ be the policy over network $\{\kappa_1, \kappa_2\}$ that maps facts $R(a, a)$ to node $\kappa_1$, for all $a \in \text{dom}$, and all other $R$-facts and all $S$-facts to node $\kappa_2$.

We argue that $Q$ is parallel-complete under $P$ on all instances. Indeed, assume $H(a, b) \in Q(I)$ for some instance $I$ and $a, b \in \text{dom}$. If $a \neq b$, only the valuation $\{y \mapsto a, z \mapsto b\}$ can derive $H(a, b)$. This means that $\{R(a, b), S(a, b)\} \subseteq I$. Furthermore, $\{R(a, b), S(a, b)\} \subseteq rfacts_P(\kappa_2)$. Hence, $H(a, b) \in Q(\text{loc-inst}_P(\kappa_2))$. If $a = b$, then $R(a, a) \in I$. So, $R(a, a) \in rfacts_P(\kappa_1)$ and $H(a, a) \in Q(\text{loc-inst}_P(\kappa_1))$. On the other hand, $Q_2$ is not parallel-complete under $P$ on instance $I = \{R(0, 0), S(0, 0)\}$. Indeed, $H(0, 0) \in Q_2(I)$ but $Q_2(\text{loc-inst}_P(\kappa_1)) = Q_2(\{R(0, 0)\}) = \emptyset$ and $Q_2(\text{loc-inst}_P(\kappa_2)) = Q_2(\{S(0, 0)\}) = \emptyset$. □
We recall from Section 3.2 that disjunctions in unions of conjunctive queries use disjoint variable sets and a valuation for \( Q \) is a valuation for exactly one disjunct. As formalised next, the notion of minimality for valuations given in Definition 4 naturally extends to \( \text{UCQ}^\forall \).

- **Definition 8.** Let \( Q = \bigcup_{i=1}^n Q_i \) be a \( \text{UCQ}^\forall \). A valuation \( V_i \) for \( Q_i \), with \( i \in \{1, \ldots, n\} \), is minimal for \( Q \), if for no \( j \in \{1, \ldots, n\} \) there is a valuation \( V_j \) for \( Q_j \), such that \( V_i(\text{head}_{Q_i}) = V_j(\text{head}_{Q_j}) \) and \( V_i(\text{body}_{Q_i}) \subseteq V_j(\text{body}_{Q_j}) \).

- **Example 9.** Consider a simple \( \text{UCQ}^\forall \) \( Q = Q_1 \cup Q_2 \) where \( Q_1, Q_2 \in \text{CQ}^\forall \) are as follows:

\[
Q_1 : \quad H(u, v) \leftarrow R(u, v), R(v, u), R(u, u),
Q_2 : \quad H(x, y) \leftarrow R(x, y), R(y, z), y \neq z.
\]

Valuation \( V_2 \overset{\text{def}}{=} \{ x \mapsto 0, y \mapsto 0, z \mapsto 1 \} \) is not minimal for \( Q \) because valuation \( V_1 \overset{\text{def}}{=} \{ u \mapsto 0, v \mapsto 0 \} \) derives the same fact \( H(0, 0) \) requiring only \( \{ R(0, 0) \} \subseteq \{ R(0, 0), R(0, 1) \} \). Similarly, valuation \( W_1 \overset{\text{def}}{=} \{ u \mapsto 0, v \mapsto 1 \} \), requiring \( \{ R(0, 1), R(1, 0), R(0, 0) \} \), is not minimal for \( Q \) because valuation \( W_2 \overset{\text{def}}{=} \{ x \mapsto 0, y \mapsto 1, z \mapsto 0 \} \) only requires \( \{ R(0, 1), R(1, 0) \} \).

The notion of minimality leads to basically the same simple characterisation of parallel-completeness:

- **Lemma 10.** A \( \text{UCQ}^\forall \) \( Q \) is parallel-correct under distribution policy \( P = (U, r\text{facts}_P) \) if and only if the following holds:

\[
\text{For every minimal valuation } V \text{ for } Q \text{ over } U, \text{ there is a node } \kappa \in \mathcal{N} \text{ such that } V(\text{body}_Q) \subseteq r\text{facts}_P(\kappa). \quad (C1')
\]

**Proof.** (If) Assume \((C1')\) holds. Because of monotonicity, we only need to show that \( Q(I) \subseteq \bigcup_{\kappa \in \mathcal{N}} Q(\text{loc-inst}_{P,I}(\kappa)) \) for every instance \( I \). To this end, let \( f \) be an arbitrary fact that is derived by some valuation \( V \) for \( Q \) on \( I \). Then, there is also a minimal valuation \( V' \) that is satisfying on \( I \) and which derives \( f \). Because of \((C1')\), there is a node \( \kappa \in \mathcal{N} \) where all facts required by \( V' \) meet (cf. Remark 6). Hence, \( f \in \bigcup_{\kappa \in \mathcal{N}} Q(\text{loc-inst}_{P,I}(\kappa)) \), i.e. query \( Q \) is parallel-correct under policy \( P \).

(Only if) For a proof by contraposition, suppose that there is a minimal valuation \( V' \) for \( Q \) for which the required facts do not meet under \( P \). Consider the input instance \( I = V'(\text{body}_Q) \). By definition of minimality, there is no valuation that agrees on the head variables and is satisfying for \( Q \) on a strict subset of \( V'(\text{body}_Q) \). Therefore, \( V'(\text{head}_Q) \) is in \( Q(I) \) but it is not derived on any node and thus query \( Q \) is not parallel-complete under policy \( P \). \( \square \)

The characterisation in Lemma 10, in turn, can be used to prove a \( \Pi^P_2 \) upper bound.

- **Lemma 11.** Parallel-Correct(\( \text{UCQ}^\forall, P \)) is in \( \Pi^P_2 \), for every \( P \in \mathcal{P} \).

**Proof.** It suffices to show that the complement of Parallel-Complete(\( \text{UCQ}^\forall, \mathcal{P}^{k,n}_{\text{poly}} \)) is in \( \Sigma^P_2 \) for arbitrary \( k \geq 2 \). Let \( P = (n, T) \) be a policy from \( \mathcal{P}^{k,n}_{\text{poly}} \). We have to consider only instances whose data values can be represented by strings of length \( n \) over networks whose nodes can be represented by strings of length \( n \).

By Lemma 10, a query \( Q \) is not parallel-correct under distribution policy \( P \) if and only if there exists a minimal valuation \( V \) that satisfies \( Q \) on some instance \( I \) with \( \text{dom}(I) \subseteq \text{dom}_n \) such that no node in \( \text{dom}_n \) is responsible for all facts from \( V(\text{body}_Q) \).

First, the algorithm non-deterministically guesses a valuation \( V \), which can be represented by a string in length polynomial in \( Q \) and \( n \). Subsequently, it checks for all valuations \( V' \),...
all nodes $\kappa$, and all strings $x$ of polynomial length whether $V'$ contradicts minimality of $V$ (in which case the algorithm rejects the input) and, by use of algorithm $T$, whether node $\kappa$ is not responsible for at least one fact from $V(\text{body}_Q)$ (if so, the algorithm continues, otherwise it rejects). All tests can be done in polynomial time.

From [4] we know the following result.

**Theorem 12 ([4]).** \textsc{Parallel-Correct}($CQ, \mathcal{P}_{fin}$) is $\Pi^p_2$-complete.

Together with Lemma 11 we get the following result.

**Theorem 13.** \textsc{Parallel-Correct}($UCQ^\#$, $\mathcal{P}$) is $\Pi^p_2$-complete, for every $\mathcal{P} \in \mathcal{P}$.

### 5 Containment of $CQ^-$ and $UCQ^-$

In this section, we establish the complexity of containment for $CQ^-$ and $UCQ^-$. We need these results to establish lower bounds on parallel-correctness and its constituents in the next section. Whereas containment for $CQ$ has been intensively studied in the literature, the analogous problems for $CQ^-$ and $UCQ^-$ have hardly been addressed and seem to belong to folklore. In fact, we only found a reference of a complexity result for containment of $CQ^-$ in [19], where a $\Pi^p_2$-algorithm for the problem is given, based on observations in [16], and the existence of a matching lower bound is mentioned. However, as we show below, although the problem is indeed in $\Pi^p_2$ for queries defined over a fixed schema (or when the arity of relations is bounded), it is $\text{coNEXPTIME}$-complete in the general case.

We first show the lower bounds. They actually already hold for Boolean queries. We show that \textsc{Containment}($BCQ^-, UBCQ^-$) is $\text{coNEXPTIME}$-hard by a reduction from the succinct 3-colorability problem and afterwards that \textsc{Containment}($BCQ^-, UBCQ^-$) can be reduced to \textsc{Containment}($BCQ^-, BCQ^-$). Here, $BCQ^-$ and $UBCQ^-$ denote the class of Boolean $CQ$’s and unions of Boolean $CQ$’s, respectively. Together this establishes that \textsc{Containment}($BCQ^-, BCQ^-$) and therefore also \textsc{Containment}($CQ^-, CQ^-$) are $\text{coNEXPTIME}$-hard.

**Proposition 14.** \textsc{Containment}($BCQ^-, UBCQ^-$) is $\text{coNEXPTIME}$-hard.

**Proof.** The proof is by a reduction from the succinct 3-colorability problem, which asks, whether a graph $G$, which is implicitly given by a circuit with binary AND- and OR- and unary NEG-gates, is 3-colorable. The latter problem is known to be $\text{NEXPTIME}$-complete [18]. We say that a circuit $C$, with $2\ell$ Boolean inputs, describes a graph $G = (N, E)$, when $N = \{0, 1\}^\ell$, and there is an edge $(n_1, n_2) \in N^2$ if and only if $C$ outputs true on input $n_1,n_2$.

Let $C$ be an input for the succinct 3-colorability problem with $2\ell$ Boolean inputs. We construct queries $Q_1$ and $Q_2$ such that $Q_1 \not\subseteq Q_2$ if and only if the graph described by $C$ is 3-colorable.

Both queries are over schema $D$, which consists of relation names $\text{DomainValues}^{(3)}$, $\text{Bool}^{(1)}$, $\text{And}^{(3)}$, $\text{Or}^{(3)}$, $\text{Neg}^{(2)}$, and $\text{Label}^{(\ell+1)}$. Intuitively, satisfaction of $Q_1$ will guarantee that there is a tuple $(a_0, a_1, a_2)$ with three different values in relation $\text{DomainValues}$. We will use, for some such tuple, $a_0,a_1,a_2$ as colors and $a_0,a_1$ as truth values. We will often assume without loss of generality that $(a_0, a_1, a_2) = (0, 1, 2)$. In particular, for such a tuple, $a_0$ is interpreted as false while $a_1$ is interpreted as true. The unary relation $\text{Bool}$ will be forced by $Q_1$ to contain at least $a_0$ and $a_1$.

Relations $\text{And}$, $\text{Or}$, and $\text{Neg}$ are intended to represent the respective logical functions. The first two attributes represent input values, and the last attribute represents the output.
Again, $Q_1$ will guarantee that at least all triples of Boolean values that are consistent with the semantics of AND, OR, and NEG are present in these relations. Tuples in relation $Label$ represent nodes together with their respective color (one can think of the representation of a node by $\ell$-ary addresses over a ternary alphabet).

We define query $Q_1$ as follows:

$$T() \leftarrow \text{DomainValues}(w_0, w_1, w_2), \neg \text{DomainValues}(w_1, w_0, w_2),$$
$$\neg \text{DomainValues}(w_2, w_1, w_0), \neg \text{DomainValues}(w_0, w_2, w_1),$$
$$\text{And}(w_0, w_0), \text{And}(w_0, w_0), \text{And}(w_0, w_0), \text{And}(w_0, w_1), \text{And}(w_1, w_1, w_1),$$
$$\text{Or}(w_0, w_0, w_0), \text{Or}(w_0, w_1, w_1), \text{Or}(w_1, w_0, w_1), \text{Or}(w_1, w_1, w_1).$$

It is easy to see that $Q_1$ enforces the conditions mentioned above.

In the following, we denote sequences $x_1, \ldots, x_\ell$ of $\ell$ variables by $x$.

We define $Q_2$ as the union of the queries $Q^1_2$ and $Q^2_2$, where subquery $Q^1_2$ is defined as:

$$T() \leftarrow \text{Bool}(x_1), \text{Bool}(x_2), \ldots, \text{Bool}(x_\ell), \text{DomainValues}(y_r, y_g, y_b),$$
$$\neg \text{Label}(x, y_r), \neg \text{Label}(x, y_g), \neg \text{Label}(x, y_b).$$

Intuitively, $Q^1_2$ can be satisfied in a database if for some node, represented by $x$, there is no color.

Subquery $Q^2_2$ deals with the correctness of a coloring and uses a set $\text{CIRCUIT}$ of atoms that is intended to check whether for two nodes $u$ and $v$, represented by $y$ and $z$, respectively, there is an edge between $u$ and $v$.

To this end, $\text{CIRCUIT}$ uses the variables $y_1, \ldots, y_\ell, z_1, \ldots, z_\ell$, representing the input and, at the same time, the $2\ell$ input gates of $C$, and an additional variable $u_i$ for each gate of $C$, with the exception of the output gate. The output gate is represented by variable $w_1$. For each AND-gate represented by variable $v_1$ with incoming edges from gates represented by variables $u_1$ and $u_2$, $\text{CIRCUIT}$ contains an atom $\text{And}(u_1, u_2, v_1)$. Likewise for OR- and NEG-gates.

Subquery $Q^2_2$ is defined as:

$$T() \leftarrow \text{DomainValues}(w_0, w_1, w_2), \text{CIRCUIT}, \text{Label}(y, u), \text{Label}(z, u).$$

Intuitively, $Q^2_2$ returns true when two nodes, witnessed to be adjacent by the circuit, have the same color.

Correctness of the reductions can be shown rather straightforwardly, as is done in the full version of this paper [14].

Next, we provide the above mentioned reduction.

\textbf{Proposition 15.} $\text{Containment}(BCQ^-, UBCQ^-) \leq_P \text{Containment}(BCQ^-, BCQ^-)$.

\textbf{Proof.} Let $Q_1$ be in $BCQ^-$ and $Q_2 = \bigcup_{i=1}^{m} Q^i_2$ be in $UBCQ^-$ over some database schema $\mathcal{D}$. Recall our assumption, that each disjunct is defined over a disjoint set of variables. Next, we construct CQs $Q'_1$ and $Q'_2$ such that $Q'_1 \subseteq Q'_2$ if and only if, $Q_1 \subseteq Q_2$.

We explain the intuition behind the reduction by means of an example. To this end, let $Q_1$ be $H() \leftarrow A(x, y)$ and let $Q_2$ be the $Q^1_2 \cup Q^2_2$, where $Q^1_2$ is $H() \leftarrow A(u_1, v_1), B(u_1, v_1)$
and \( \mathcal{Q}_2^\alpha \) is \( H() \leftarrow A(u_2, v_2), \neg B(u_2, v_2) \), both formulated over the schema \( \mathcal{D} = \{ A^{(2)}, B^{(2)} \} \).

The query \( \mathcal{Q}_2^\alpha \) takes the following form:

\[
H() \leftarrow \text{Active}(x_0, x_1; \ell_1, \ell_2), \alpha(\ell_1, \mathcal{Q}_{2,1}^\alpha), \alpha(\ell_2, \mathcal{Q}_{2,2}^\alpha)\]

where \( \alpha(w, \mathcal{Q}) \) denotes the modification of the body of \( \mathcal{Q} \) by replacing every atom \( R(x) \) by \( R'(w, x) \). Both queries are defined over the schema \( \mathcal{D}' = \{ A^{(3)}, B^{(3)}, \text{Active}(4) \} \). Notice that \( \mathcal{Q}_2^\alpha \) contains a concatenation of the disjuncts of \( \mathcal{Q}_2 \). In addition, relations \( A \) and \( B \) are extended with a new first column with the purpose of labelling tuples. This labelling allows to encode two (or even more) instances over \( \mathcal{D} \) by one instance over \( \mathcal{D}' \). Specifically, \( \text{body}_{\mathcal{D}'} \) (not shown) is constructed in such a way that when there is a satisfying valuation for \( \mathcal{Q}_2 \) there are two different data values, say 0 and 1. So, an instance \( I \) over \( \mathcal{D} \) can be encoded as \( I^0 = \{ A'(0, a, b) \mid A(a, b) \in I \} \cup \{ B'(0, a, b) \mid B(a, b) \in I \} \) or as \( I^1 = \{ A'(1, a, b) \mid A(a, b) \in I \} \cup \{ B'(1, a, b) \mid B(a, b) \in I \} \). In addition, when there is a satisfying valuation for \( \mathcal{Q}_2^\alpha \), there is an instance \( I_2 \) on which every disjunct of \( \mathcal{Q}_2 \) is true, and there is an instance \( I_1 \) on which \( \mathcal{Q}_1 \) is true. So, both \( \mathcal{Q}_2^\alpha \) and \( \mathcal{Q}_2^\alpha \) evaluate to true on \( I_2^0 \) when \( \ell_1 \) and \( \ell_2 \) are interpreted by label 0. However, for \( \mathcal{Q}_1 \) to be contained in \( \mathcal{Q}_2 \), we need that at least one of the disjuncts \( \mathcal{Q}_2^\alpha \) or \( \mathcal{Q}_2^\alpha \) evaluates to true over \( I_1^1 \), that is, when its labelling variable is interpreted as 1. Atom \( \text{Active}(x_0, x_1; \ell_1, \ell_2) \) will ensure that \( x_0 \) and \( x_1 \) correspond with the values 0 and 1, and that at least one of the labelling variables \( \ell_1 \) or \( \ell_2 \) is equal to 1. In other words, \( \text{Active} \) chooses which disjunct to activate over \( I_1 \). So, at least one disjunct of \( \mathcal{Q}_2 \) evaluates to true on the instance \( I_1 \) on which \( \mathcal{Q}_1 \) is satisfied.

The reduction is explained in more detail in the full version of this paper [14].

Combining Propositions 14 and 15 we get the following corollary:

**Corollary 16.** \( \text{CONTAINMENT}(\mathcal{CQ}^-, \mathcal{CQ}^-) \) is \( \text{coNEXPTIME-hard} \).

The corresponding upper bounds hold also in the presence of inequalities and are shown by small model (i.e., counter-example) properties. To this end, we make use of a restricted monotonicity property of \( \mathcal{UCQ}^{-\#} \) which was already observed in Proposition 2.4 of [1]. For an instance \( I \) and a set \( D \) of data values we denote by \( I_D \) the restriction of \( I \) to facts that only use values from \( D \).

**Lemma 17 ([1]).** For \( \mathcal{Q} \in \mathcal{UCQ}^{-\#} \), \( I \) an instance with a compatible schema, and \( D \) a set of data values, it holds that \( \mathcal{Q}(I_D) \subseteq \mathcal{Q}(I) \).

**Proof.** Let \( f \in \mathcal{Q}(I_D) \) via a valuation \( V \) for a disjunct \( \mathcal{Q}_i \) of \( \mathcal{Q} \). Thus, \( V(\text{pos}_{\mathcal{Q}_i}) \subseteq I_D \subseteq I \).

By definition, every variable \( x \) of \( \mathcal{Q}_i \) occurs in a positive atom and therefore \( V(x) \in D \). Thus, \( V(\text{neg}_{\mathcal{Q}_i}) \cap I = V(\text{neg}_{\mathcal{Q}_i}) \cap I_D = 0 \) and \( f \in \mathcal{Q}(I) \) as claimed.

Now we can establish the following small model property for testing containment.

**Lemma 18.** Let \( \mathcal{Q}_1, \mathcal{Q}_2 \in \mathcal{UCQ}^{-\#} \). If there is an instance \( I \), where \( \mathcal{Q}_1(I) \nsubseteq \mathcal{Q}_2(I) \), then there is also an instance \( J \subseteq I \), where \( \mathcal{Q}_1(J) \nsubseteq \mathcal{Q}_2(J) \), and \( |\text{adom}(J)| \leq |\text{varmax}(\mathcal{Q}_1)| \).

**Proof.** Let \( I \) be as in the lemma and let \( f \) be a fact with \( f \in \mathcal{Q}_1(I) \) and \( f \notin \mathcal{Q}_2(I) \). Let \( V \) be a valuation that derives \( f \) via some disjunct \( \mathcal{Q}_1^i \) of \( \mathcal{Q}_1 \).

Let \( D \equiv \text{adom}(V(\text{pos}_{\mathcal{Q}_1})) \) and \( J \equiv I_D \) the set of all facts in \( I \) using only values from \( \text{adom}(V(\text{pos}_{\mathcal{Q}_1})) \). By definition, \( |\text{adom}(J)| \leq |\text{varmax}(\mathcal{Q}_1)| \). Clearly, \( V \) is still a satisfying valuation for \( \mathcal{Q}_1^i \) over \( J \). However, by Lemma 17, \( f \notin \mathcal{Q}_2(J) \equiv \mathcal{Q}_2(I_D) \).
The upper bounds follow easily from Lemma 18.

**Proposition 19.** The following upper bounds hold:
1. Containment($UCQ^{\neg \varphi}, UCQ^{\neg \varphi}$) is in coNEXPTIME.
2. For every $k$, containment of $UCQ^{\neg \varphi}$-queries over schemas with arity bound $k$ is in $\Pi^p_2$.

**Proof.** In both cases, we consider the complement of Containment($UCQ^{\neg \varphi}, UCQ^{\neg \varphi}$).

1. A NEXPTIME algorithm, on input $Q_1, Q_2$, can simply guess an instance $J$ with a domain of at most $m$ elements and a fact $f$, and verifies that $f \in Q_1(J)$ but $f \notin Q_2(J)$. For the latter tests, it can simply cycle, in exponential time, to all valuations over $J$ for $Q_1$ and $Q_2$.

2. For a fixed arity bound, the minimal counter-example $J$ is of size at most $m^k$. It can thus be guessed in polynomial time. That $f \in Q_1(J)$ can be verified non-deterministically. That $f \notin Q_2(J)$ can be verified by a universal computation in polynomial time. ▶

A claim of a $\Pi^p_2$ upper bound for containment of CQs with negation can be found in [19]. It was not made clear there, that this claim assumes bounded arity of the schema. That the containment problem is $\Pi^p_2$-complete for schemas of bounded arity has been explicitly shown in [17]. Clearly, Proposition 19.2 follows directly and 19.1 is only a variation of it. From Proposition 19 and Corollary 16 the main result of this section immediately follows.

**Theorem 20.** Containment($BCQ^-, UCQ^{\neg \varphi}$) and Containment($UCQ^{\neg \varphi}, UCQ^{\neg \varphi}$) are coNEXPTIME-complete.

Of course, the theorem also holds for all classes $C$ of queries with $BCQ^- \subseteq C \subseteq UCQ^{\neg \varphi}$.

## 6 Parallel-correctness: unions of conjunctive queries with negation

As mentioned in Section 4, for conjunctive queries without negation parallel-soundness always holds and thus parallel-correctness and parallel-completeness coincide, thanks to monotonicity. For queries with negation the situation is different. Distributed evaluation can be complete but not sound, or vice versa. For this reason, we have to distinguish all three problems separately: correctness, soundness, and completeness. However, the complexity is the same in all three cases.

Our results show a second, more crucial difference. Whereas parallel completeness for CQs without negation could be characterised in terms of valuations, that is, objects of polynomial size, our algorithms for CQs with negation involve counter-examples of exponential size (if the arity of schemas is not bounded) and the coNEXPTIME lower bound results indicate that this is unavoidable. We illustrate the observation that counter-examples might need an exponential number of tuples by the following example.

**Example 21.** Let $Q$ be the following conjunctive query with negation:

$$H() \leftarrow \text{Bool}(w_0, w_0), \text{Bool}(w_1, w_1), \text{Bool}(x_1, x_1), \ldots, \text{Bool}(x_n, x_n),$$

$$\neg\text{Bool}(w_0, w_1), \neg\text{Rel}(x_1, \ldots, x_n).$$

Let $P$ be the policy defined over universe $U = \{0, 1\}$ and two-node network $\{\kappa_1, \kappa_2\}$, which distributes all facts except $\text{Rel}(0, \ldots, 0)$ to node $\kappa_1$ and only fact $\text{Rel}(0, \ldots, 0)$ to node $\kappa_2$.

Query $Q$ is not parallel-sound under policy $P$, but the smallest counter-example $I$ is of exponential size as we argue next. Indeed, let $I \equiv \{\text{Bool}(0, 0), \text{Bool}(1, 1)\} \cup \{\text{Rel}(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in \{0, 1\}^n\}$. Furthermore, let valuation $V$ map variables $w_1$ and $w_0$ to 1 and 0,
are in distribution policies. If the arity of schemas is bounded by some fixed number, these problems are
Parallel-Correct.

Towards (1) let us assume that

\[ \{ \text{Rel}(0,\ldots,0) \} \]

because neither \text{Bool}(0,1) nor \text{Rel}(0,\ldots,0) is contained in the local instance. Furthermore, there is no satisfying valuation \( W \) for \( Q \) on the global instance \( I \) because \( W \) would have to map each \( x_i \) to either 0 or 1 implying that

\[ W(\text{Rel}(x_1,\ldots,x_n)) \in I. \]

However, there is no smaller instance: let \( I^* \) be some instance over universe \( U \) that has a locally satisfying valuation \( V \). The combination of atoms \( \text{Bool}(w_0,0), \text{Bool}(w_1,0) \), and \( \neg \text{Bool}(0,1) \) in query \( Q \) then implies existence of both facts \( \text{Bool}(0,0) \) and \( \text{Bool}(1,1) \) because variables \( w_0 \) and \( w_1 \) cannot be mapped onto the same data value.

Assume that fact \( \text{Rel}(a_1,\ldots,a_n) \), for some \( (a_1,\ldots,a_n) \in \{0,1\}^n \) is missing from \( I^* \). Then the valuation \( W \) that maps \( w_0 \mapsto 0, w_1 \mapsto 1 \) and \( x_i \mapsto a_i \), for every \( i \in \{1,\ldots,n\} \), satisfies \( Q \) also globally, on instance \( I^* \), and can therefore be no example against parallel-soundness, which contradicts our choice of \( I^* \). Thus, \( \text{Rel}(a_1,\ldots,a_n) \in I^* \), for every \( (a_1,\ldots,a_n) \in \{0,1\}^n \). We therefore have \( I \subseteq I^* \) and, in particular, instance \( I^* \) contains at least as many facts as instance \( I \).

The results of this section are summarised in the following theorem:

**Theorem 22.** For every class \( P \in \{ P_{\text{rule}} \} \cup \Psi_{\text{npoly}} \) of distribution policies, the following problems are \( \text{coNEXPTIME} \)-complete.

- Parallel-Sound(UCQ, \( P \))
- Parallel-Complete(UCQ, \( P \))
- Parallel-Correct(UCQ, \( P \))

Theorem 22 follow from Propositions 23 and 25 below. It also holds for UCQ\(^{-\#}\). It is easy to show that, when restricted to schemas with some fixed (but sufficiently large, for hardness) arity bound, all these problems are \( \Pi_2 \)-complete.

### 6.1 Upper bounds

In this section, we show the upper bounds of Theorem 22, summarised in the following proposition.

**Proposition 23.** Parallel-Sound(UCQ\(^{-\#}\), \( P \)), Parallel-Complete(UCQ\(^{-\#}\), \( P \)), and Parallel-Correct(UCQ\(^{-\#}\), \( P \)) are in \( \text{coNEXPTIME} \), for every class \( P \in \Psi \) of distribution policies. If the arity of schemas is bounded by some fixed number, these problems are in \( \Pi_2 \).

**Proof.** As already indicated above, the proof relies on a bound on the size of a smallest counter-example. More specifically, we first show the following claim.

**Claim 24.** Let \( Q \in \text{UCQ\(^{-\#}\)} \) and let \( P \) be an arbitrary distribution policy. Then the following statements hold:

1. If \( Q \) is not parallel-complete under \( P \), then there is an instance \( J \) over a domain with at most varmax(\( Q \)) elements such that \( Q \) is not parallel-complete on \( J \) under \( P \).
2. If \( Q \) is not parallel-sound under \( P \), then there is an instance \( J \) over a domain with at most varmax(\( Q \)) elements such that \( Q \) is not parallel-sound on \( J \) under \( P \).

Towards (1) let us assume that \( Q \) is not parallel-complete on some instance \( I \) under \( P \). Let \( V \) be a valuation of a disjunct \( Q_i \) of \( Q \) that derives a fact \( f \) globally that is not derived on any node of the network. Let \( D \equiv \text{adom}(V(\text{pos}_Q)) \) and \( J \equiv I_D \). Clearly, \( |D| \leq \text{varmax}(Q) \) and \( V \) still derives \( f \) globally on instance \( J \) via \( Q_i \). On the other hand, for every node \( \kappa \), \( Q(\text{loc-inst}_{P,I}(\kappa)) = Q(\text{loc-inst}_{P,I}(\kappa)_{|D}) \subseteq Q(\text{loc-inst}_{P,I}(\kappa)) \), thanks to
Lemma 17. Therefore \( f \) is not derived on \( \kappa \), and thus \( J \) witnesses the lack of parallel-completeness of \( Q \) under \( P \).

The proof of (2) is completely analogous. Given a counter-example \( I \) and a valuation \( V \) that derives a fact \( f \) on some node \( \kappa \) via \( Q \), for which \( f \) is not derived globally, we define \( D \overset{\text{def}}{=} I_{\text{dom}(V(\text{pos}_Q))} \) and show that \( J \overset{\text{def}}{=} I_D \) is the desired counter-example.

An algorithm that tests the complement of parallel-completeness non-deterministically is described in the full version of this paper [14]. ▶

6.2 Lower bounds

The lower bounds stated in Theorem 22 follow from a polynomial time reduction from problem \( \text{CONTAINMENT}(\text{BCQ}^- \cup \text{BCQ}^+) \), for which we showed \( \text{coNEXPTIME} \)-hardness in Section 5.

▶ Proposition 25. \( \text{PARALLEL-COMPLETE}(\text{CQ}^- \cup \text{P}_{\text{rule}}) \), \( \text{PARALLEL-SOUND}(\text{CQ}^- \cup \text{P}_{\text{rule}}) \), and \( \text{PARALLEL-CORRECT}(\text{CQ}^- \cup \text{P}_{\text{rule}}) \) are \( \text{coNEXPTIME} \)-hard.

Proof. Interestingly, all three results are shown by the same reduction from decision problem \( \text{CONTAINMENT}(\text{BCQ}^- \cup \text{BCQ}^+) \).

The basic idea for this reduction is very simple: it combines both queries \( Q_1, Q_2 \in \text{BCQ}^- \) of the given containment instance into a single query \( Q \in \text{BCQ}^- \) and infers an appropriate distribution policy \( P \). To emulate separate derivation for both queries in the combined query, an activation mechanism is used that resembles the proof of Proposition 15. In this fashion, the two queries can be evaluated over different subsets of the considered instance by annotating both the facts in the instance as well as the atoms of the query.

We next describe the reduction in detail. Let thus \( Q_1, Q_2 \in \text{BCQ}^- \) be queries over some schema \( D \) and let \( m \overset{\text{def}}{=} \max \{ \text{varmax}(Q_1), \text{varmax}(Q_2) \} \). Without loss of generality, we assume the variable sets of \( Q_1 \) and \( Q_2 \) to be disjoint. We will also assume in the following that both \( Q_1 \) and \( Q_2 \) are satisfiable. This is the case (for \( Q_1 \)) if and only if \( \text{pos}_{Q_1} \cap \text{neg}_{Q_1} = \emptyset \) and can therefore be easily tested in polynomial time. If one of the test fails, some appropriate constant instance of \( \text{PARALLEL-COMPLETE}(\text{CQ}^- \cup \text{P}_{\text{rule}}) \) or one of the other problem variants, respectively, can be computed.

We define a (Boolean) query \( Q \in \text{BCQ}^- \) and a policy \( P \in \text{P}_{\text{rule}} \) over domain \( \{1, \ldots, m\} \) that can be computed from \( Q_1 \) and \( Q_2 \) in polynomial time. The schema for \( Q \) is \( \mathcal{D}' \overset{\text{def}}{=} \{ R_{k+1}^{(k)} \mid R_k \in \mathcal{D} \} \). That is, each relation name \( R \) of \( \mathcal{D} \) occurs as \( R' \in \mathcal{D}' \) with an arity incremented by one. Additionally, \( Q \) uses relation names \( \text{Type}, \text{Start}_1, \text{Start}_2, \) and \( \text{Stop} \), which we assume not to occur in schema \( D \). Besides the variables of \( Q_1 \) and \( Q_2 \), query \( Q \) uses variables \( t_1, t_2, t \).

We use the function \( \alpha \), defined in the proof of Proposition 15, which adds its first parameter as first component to every tuple in its second parameter and translates relation names \( R \) into \( R' \). In Proposition 15, the first parameter was always a variable and the second a set of atoms, but we use \( \alpha \) also for a data value as first and a set of facts as second parameter in the obvious way. We write \( \alpha_a^{-1} \) for the function mapping sets of facts over \( \mathcal{D}' \) to sets of facts over \( \mathcal{D} \), by selecting, from a set of facts, all facts with first parameter \( a \), deleting this parameter and replacing each name \( R' \) by \( R \). Finally, \( \pi_a(I) \overset{\text{def}}{=} \alpha(a, \alpha_a^{-1}(I)) \) is the restriction of \( I \) to all facts with \( a \) in their first component.

The combined query \( Q \) has \( \text{head}_Q \overset{\text{def}}{=} H() \) and body

\[
\text{body}_Q \overset{\text{def}}{=} \alpha(t_1, \text{body}_{Q_1}) \cup \alpha(t_2, \text{body}_{Q_2}) \cup \{ \text{Type}, \text{Start}_1(t_1), \text{Start}_2(t_1), \text{Start}_3(t_2), \text{Stop}(t_1), \lnot \text{Stop}(t_2) \}.
\]

\( A \)
Policy \( P \) is defined over universe \( U \overset{\text{def}}{=} \{1, \ldots, m\} \), schema \( D' \cup \{\text{Type}, \text{Start}_1, \text{Start}_2, \text{Stop}\} \) and network \( N \overset{\text{def}}{=} \{\kappa_1, \ldots, \kappa_m, \sigma_1, \ldots, \sigma_m, \rho\} \). Facts are distributed as follows:

- Every node \( \kappa_i \) is responsible for the facts \( \text{Type}(1), \text{Start}_1(i), \text{Start}_2(i), \text{Stop}(i) \), and all facts from \( \text{facts}(D', U) \).
- Every node \( \sigma_i \) is responsible for the facts \( \text{Type}(2), \text{Start}_1(i), \text{Stop}(i) \), all \( \text{Start}_2 \)-facts, and all facts from \( \text{facts}(D', U) \).
- Finally, node \( \rho \) is responsible for facts \( \text{Type}(3), \ldots, \text{Type}(m) \), and all facts over other relation names.

It is easy to see that \( P \) can be expressed by a polynomial number of rules and that \( Q \) and \( P \) can be computed in polynomial time. In the full version of this paper [14], we show that the described function is indeed the desired reduction.

\section{Full conjunctive queries}

In this section, we focus attention on full conjunctive queries, in an attempt to lower the complexity of testing parallel-correctness. Requiring queries to be full is a very natural restriction which is known to have practical benefits. For example, the Hypercube algorithm, which describes an optimal way to compute CQs in a setting very similar to ours, completely ignores projections when shuffling data, and only applies them when computing the query locally. The latter is possible because correctness for the full-variant of a query is in a sense more strict than correctness for the query itself.

Formally, a (union of) conjunctive queries is called \emph{full} if all variables of the body also occur in the head. We denote by \( \text{FCQ}^- \) and \( \text{UFCQ}^- \) the class of full CQ\(^-\) and full UCQ\(^-\) queries, respectively, and likewise for other fragments.

The presentation is similar to that of Section 5 and 6. First, we establish the complexity of query containment. Then, we show that containment reduces to parallel-correctness (and variants). Finally, we obtain matching upper bounds.

The following theorem shows that unlike for general conjunctive queries the complexity of deciding containment for \( \text{FCQ}^- \) and \( \text{UFCQ}^- \) do not coincide.

\begin{theorem}
\begin{enumerate}
\item \( \text{Containment}(\text{FCQ}^-, \text{FCQ}^-) \) is in \( P \);
\item \( \text{Containment}(\text{FCQ}^-, \text{UFCQ}^-) \) is \( \text{coNP}\)-complete; and
\item \( \text{Containment}(\text{UFCQ}^-, \text{UFCQ}^-) \) is \( \text{coNP}\)-complete.
\end{enumerate}
\end{theorem}

As one can reduce from \( \text{Containment}(\text{FCQ}^-, \text{UFCQ}^-) \) to parallel-soundness, completeness, and correctness, we obtain the following hardness results:

\begin{proposition}
\begin{enumerate}
\item \( \text{Parallel-Sound}(\text{UFCQ}^-, \mathcal{P}) \), \( \text{Parallel-Complete}(\text{UFCQ}^-, \mathcal{P}) \), and \( \text{Parallel-Correct}(\text{UFCQ}^-, \mathcal{P}) \) are \( \text{coNP}\)-hard, for every \( \mathcal{P} \in \{\mathcal{P}_{\text{rule}}\} \cup \mathcal{P}_{\text{npoly}} \).
\end{enumerate}
\end{proposition}

The following theorem determines the complexity for the upper bounds:

\begin{theorem}
The following problems are \( \text{coNP}\)-complete:
\begin{enumerate}
\item \( \text{Parallel-Sound}(\text{UFCQ}^-, \mathcal{P}_{\text{rule}}) \);
\item \( \text{Parallel-Complete}(\text{UFCQ}^-, \mathcal{P}_{\text{rule}}) \);
\item \( \text{Parallel-Correct}(\text{UFCQ}^-, \mathcal{P}_{\text{rule}}) \).
\end{enumerate}
\end{theorem}
In this paper, we continued the study of parallel-correctness initiated by Ameloot et al. [4] as a framework for reasoning about one-round evaluation algorithms for conjunctive queries under arbitrary distribution policies. Specifically, we considered the case with union and negation. While parallel-correctness for unions of conjunctive queries can be tested by examining properties of single valuations, just like in the union-free case, the latter no longer holds true when negation is present. Consequently, we obtained that deciding parallel-correctness for unions of conjunctive queries remains in $\Pi^p_2$, while the analog problem in the presence of negation is hard for coNEXPTIME. Since conjunctive queries with negation are no longer monotone, we considered the related problems of parallel-completeness and parallel-soundness as well and obtained the same bounds. Interestingly, when negation is present, containment of conjunctive queries can be reduced to parallel-correctness (and its variants) allowing the transfer of lower bounds. We prove that containment for conjunctive queries with negation is hard for coNEXPTIME, which, to the best of our knowledge, is a novel result. In an attempt to lower complexity, we show that parallel-correctness for unions of full conjunctive queries with negation is coNP-complete.

There are quite a number of directions towards future work. While parallel-correctness for first-order logic is undecidable, it would be interesting to determine the exact frontier for decidability. As the considered problem is a static analysis problem that relates to the size of the queries and not to the size of the instances (at least in the setting of $P_{\text{rule}}$), exponential lower bounds do not necessarily exclude practical application. It could still be interesting to identify settings that would make parallel-correctness tractable. Possibly independent of tractability considerations, such settings could incorporate bag semantics, integrity constraints, or specific classes (and representations) of distribution policies. We also plan to consider evaluation algorithms that use knowledge about the distribution policy to compute better query results, locally. Another direction for future work is to investigate transferability of parallel-correctness for conjunctive queries as defined in [4] in the presence of union and negation.

References


