FedQPL: A Language for Logical Query Plans over Heterogeneous Federations of RDF Data Sources

(Extended Version)*

Sijin Cheng
sijin.cheng@liu.se
Dept. of Computer and Information Science (IDA),
Linköping University
Linköping, Sweden

Olaf Hartig
olaf.hartig@liu.se
Dept. of Computer and Information Science (IDA),
Linköping University
Linköping, Sweden

ABSTRACT

Federations of RDF data sources provide great potential when queried for answers and insights that cannot be obtained from one data source alone. A challenge for planning the execution of queries over such a federation is that the federation may be heterogeneous in terms of the types of data access interfaces provided by the federation members. This challenge has not received much attention in the literature. This paper provides a solid formal foundation for future approaches that aim to address this challenge. Our main conceptual contribution is a formal language for representing query execution plans; additionally, we identify a fragment of this language that can be used to capture the result of selecting relevant data sources for different parts of a given query. As technical contributions, we show that this fragment is more expressive than what is supported by existing source selection approaches, which effectively highlights an inherent limitation of these approaches. Moreover, we show that the source selection problem is NP-hard and in Σ_2^{P}, and we provide a comprehensive set of rewriting rules that can be used as a basis for query optimization.

CCS CONCEPTS

• **Information systems** → **Federated databases**; **Query planning**; *Mediators and data integration*; *Web interfaces*.

KEYWORDS

Federation, RDF, SPARQL, Linked Data Fragments

1 INTRODUCTION

Existing research on querying federations of RDF data sources focuses on federations that are homogeneous in terms of the type of interface via which each of the federation members can be accessed. In particular, the majority of work in this context assumes that all federation members provide the SPARQL endpoint interface (e.g., [3, 7, 22, 23, 25, 28]), whereas another line of research focuses solely on URI lookups (e.g., [13, 17, 26]). However, there exist other types of Web interfaces to access an RDF data source, including the Triple Pattern Fragment (TPF) interface [27], the Bindings-Restricted TPF (brTPF) interface [14], the SaGe interface [18], and the smart-KG interface [4]. Given the fact that each type of interface has particular properties and makes different tradeoffs [4, 14, 15, 18, 27] (and the same will hold for other types of such interfaces proposed in the future), any provider of an RDF data source may choose to offer a different type of interface, which leads to federations that are heterogeneous in terms of these interfaces.

Such a heterogeneity poses extra challenges for query federation engines—especially during query planning—because different interfaces may require (or enable!) the engine to leverage specific physical operators, and not all forms of subqueries can be answered directly by every interface. To the best of our knowledge, there does not exist any research on systematic approaches to tackle these challenges. This paper provides a starting point for such research.

We argue that any principled approach to query such heterogeneous federations of RDF data sources has to be based on a solid formal foundation. This foundation should provide not only a formal data model that captures this notion of federations, including a corresponding query semantics, but also formal concepts to precisely define the artifacts produced by the various steps of query planning. There are typically three main types of such artifacts in a query federation engine: the results of the query decomposition & source selection step [28], logical query plans, and physical query plans. We observe that, so far, such artifacts have been treated very informally in the literature on query engines for (homogeneous) federations of RDF data. That is, the authors talk about query plans only in terms of examples, where these examples are typically informal illustrations that visually represent some form of a tree in which the leaf nodes are one or more triple patterns with various annotations and the internal nodes are operators from the SPARQL algebra (sometimes in combination with some additional, informally-introduced operators [25]). The lack of approaches to describe query plans formally is the focus of our work in this paper.

Contributions and organization of the paper: After introducing a suitable formal data model and a corresponding query semantics for using the query language SPARQL (cf. Section 3), we make our main conceptual contribution: We define a language, called *FedQPL*, that can be used to describe logical query plans formally (cf. Section 4). This language can be applied both to define query planning and optimization approaches in a more precise manner and to actually represent the logical plans in a query engine. FedQPL features operators to explicitly capture the intention to execute a particular subquery at a specific federation member and to distinguish whether such an access to a federation member is meant to be based solely on the given subquery or also on intermediate results obtained for other subqueries. We argue that such

*This manuscript is an extended version of a paper in the 22nd International Conference on Information Integration and Web-based Applications & Services (iiWAS2020). The difference to the conference version is that this extended version includes an appendix with the full proofs of the results in the paper (see page 11).

features are paramount for any principled approach to query planning in heterogeneous federations where the characteristics and limitations of different data access interfaces have to be taken into account (and, certainly, these features can also be leveraged when defining new approaches that focus on homogeneous federations).

Given the full definition of FedQPL, we then study a specific fragment of this language that can be used to describe which federation members have to be contacted for which part of a given query (cf. Section 5). Hence, this language fragment provides a formal tool that is needed to develop well-defined source selection approaches. Moreover, in practice, this language fragment can be used to represent the results of such approaches directly as *initial logical query plans* that can then be optimized in subsequent query planning steps. As technical contributions regarding this language fragment we show that the corresponding source selection problem is NP-hard and in Σ_2^{P}. Additionally, we show that the language fragment cannot only capture any possible output of existing source selection approaches for homogeneous federations but, even when used only for such federations, it can express solutions to the source selection problem that these approaches are not able to produce.

Finally, as another technical contribution, we show a comprehensive set of equivalences for FedQPL expressions that can be used as query rewriting rules for query optimization (cf. Section 6).

Limitations: It is not the purpose of this paper to develop concrete approaches or techniques for source selection, query planning, or query optimization. Instead, we focus on providing a solid theoretical foundation for such work in the future. A specific limitation regarding our contributions in this paper is that the given definition of FedQPL covers only the join-union fragment of SPARQL. However, the formalism can easily be extended with additional operators, which is part of our future work.

2 PRELIMINARIES

This section provides a brief introduction of the concepts of RDF and SPARQL that are relevant for our work in this paper. Due to space limitations, we focus on notation and relevant symbols, and refer to the literature for the detailed formal definitions [12, 20]. It is important to note that, in this paper, we focus on the set-based semantics of SPARQL as introduced by Pérez et al. [20].

We assume pairwise disjoint, countably infinite sets: $\mathcal{U}$ (URIs), $\mathcal{B}$ (blank nodes), $\mathcal{L}$ (literals), and $\mathcal{V}$ (variables). An *RDF triple* is a tuple $(s, p, o) \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. A set of such triples is called an *RDF graph*.

The fundamental building block of SPARQL queries is the notion of a *basic graph pattern* (*BGP*) [12]; each such BGP is a nonempty set of so-called *triple patterns* where every triple pattern is a tuple in $(\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U} \cup \mathcal{L})$.[1] Other types of graph patterns for SPARQL queries can be constructed by combining BGPs using various operators (e.g., UNION, FILTER, OPTIONAL) [12].

The result of evaluating any such graph pattern P over an RDF graph G is a set—typically denoted by $[\![P]\!]_G$ [20]—that consists of so-called solution mappings, where a *solution mapping* is a partial function $\mu : \mathcal{V} \to \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$.

[1] For the sake of simplicity we do not permit blank nodes in triple patterns. In practice, each blank node in a SPARQL query can be replaced by a new variable.

For the case that P is a BGP B, $[\![P]\!]_G$ consists of every solution mapping μ for which $\mathrm{dom}(\mu) = \mathrm{vars}(B)$ and $\mu[B] \subseteq G$, where $\mathrm{vars}(B)$ is the set of all variables in B and $\mu[B]$ denotes the BGP that we obtain by replacing the variables in B according to μ; notice that $\mu[B]$ is a set of RDF triples if $\mathrm{vars}(B) \subseteq \mathrm{dom}(\mu)$. For any triple pattern tp, we write $[\![tp]\!]_G$ as a shorthand notation for $[\![\{tp\}]\!]_G$.

For any more complex graph pattern P, $[\![P]\!]_G$ is defined based on an algebra [20]. This algebra—which we call the *SPARQL algebra*—consists of several operators over sets of solution mappings, including $\bowtie$ (join) and $\cup$ (union) [20]. For instance, the join of two sets of solution mappings, Ω_1 and Ω_2, is defined as $\Omega_1 \bowtie \Omega_2 := \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2, \text{ and } \mu_1 \sim \mu_2\}$, where we write $\mu_1 \sim \mu_2$ if $\mu_1(?v) = \mu_2(?v)$ for every variable $?v$ in $\mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$; in this case, the combination of μ_1 and μ_2, denoted by $\mu_1 \cup \mu_2$, is also a solution mapping. It is not difficult to see that, for two BGPs B_1 and B_2, it holds that $[\![B_1]\!]_G \bowtie [\![B_2]\!]_G = [\![B]\!]_G$ where $B = B_1 \cup B_2$ [20]. Since the operators $\bowtie$ and $\cup$ are associative and commutative [20, 24], we can avoid parenthesis when combining more than two sets of solution mappings using either of these operators; e.g., $(\Omega_1 \bowtie \Omega_2) \bowtie \Omega_3 = \Omega_1 \bowtie \Omega_2 \bowtie \Omega_3$.

3 DATA MODEL

This section introduces a data model that captures the notion of a federation of RDF data sources that is heterogeneous in terms of the types of data access interfaces of the federation members. The model includes a query semantics that defines the expected result of executing a SPARQL query over such a federation.

A key concept of the data model is that of an *interface*, which we abstract by two components: a language for expressing data access requests and a function that, given an RDF graph, turns any expression in the language of the interface into a set of solution mappings.

Definition 1. An **RDF data access interface** (**interface**) I is a tuple $(L_{\mathrm{req}}, \varrho)$ where L_{req} denotes a language and ϱ is a function that maps every pair (ρ, G), consisting of an expression ρ in L_{req} and an RDF graph G, to a set of solution mappings.

The following three examples illustrate how any concrete interface can be defined in the context of our formalization.

Example 1. The *SPARQL endpoint interface* [8] enables clients to request the result for *any* SPARQL query over the server-side dataset. Hence, a server that provides this interface executes each requested SPARQL query over its dataset and, then, returns the result to the client. We may abstract this functionality by defining an interface $I_{\mathrm{sparql}} = (L_{\mathrm{sparqlreq}}, \varrho_{\mathrm{sparql}})$ where the expressions in $L_{\mathrm{sparqlreq}}$ are all SPARQL graph patterns and $\varrho_{\mathrm{sparql}}$ is defined for every graph pattern P and every RDF graph G such that $\varrho_{\mathrm{sparql}}(P, G) := [\![P]\!]_G$.

Example 2. While a SPARQL endpoint server enables clients to query its dataset by using the full expressive power of SPARQL, providing such a comparably complex functionality may easily overload such servers [6]. To address this issue, several more restricted types of interfaces have been proposed with the goal to shift some of the query execution effort from the server to the clients. The first of these alternatives has been the *Triple Pattern Fragment (TPF) interface* [27] that allows clients only to send triple pattern queries to the server. More specifically, via this interface, clients can request the triples from the server-side dataset that match a given

triple pattern. In terms of our model, we abstract this functionality by an interface $I_{\mathsf{TPF}} = (L_{\mathsf{TPF\text{-}req}}, \varrho_{\mathsf{TPF}})$ where the expressions in $L_{\mathsf{TPF\text{-}req}}$ are all triple patterns and ϱ_{TPF} is defined for every triple pattern tp and every RDF graph G such that $\varrho_{\mathsf{TPF}}(tp, G) := [\![tp]\!]_G$.

Example 3. The *Bindings-Restricted TPF (brTPF) interface* [14] extends the TPF interface by allowing clients to optionally attach intermediate results to triple pattern requests. The response to such a request is expected to contain only those matching triples that are guaranteed to contribute in a join with the given intermediate result. We define a corresponding interface $I_{\mathsf{brTPF}} = (L_{\mathsf{brTPF\text{-}req}}, \varrho_{\mathsf{brTPF}})$ where the expressions in $L_{\mathsf{brTPF\text{-}req}}$ are i) all triple patterns and ii) all pairs consisting of a triple pattern and a set of solution mappings, and ϱ_{brTPF} is defined for every triple pattern tp, every set Ω of solution mappings, and every RDF graph G such that

$$\varrho_{\mathsf{brTPF}}\big(tp, G\big) := [\![tp]\!]_G, \text{ and}$$

$$\varrho_{\mathsf{brTPF}}\big((tp, \Omega), G\big) := \begin{cases} [\![tp]\!]_G & \text{if } \Omega = \emptyset, \\ \{\mu \in [\![tp]\!]_G \mid \exists \mu' \in \Omega : \mu \sim \mu'\} & \text{else.} \end{cases}$$

Given the notion of an interface, we can now define our notion of a federation.

Definition 2. A **federation member** fm is a pair (G, I) that consists of an RDF graph G and an interface I. A **federation** F is a finite and nonempty set of federation members such that every member $fm = (G, I)$ in F uses a disjoint set of blank nodes; i.e., $\mathrm{bnodes}(G) \cap \mathrm{bnodes}(G') = \emptyset$ for every other $fm' = (G', I')$ in F.

Example 4. As a running example for this paper, we consider a simple federation $F_{\mathsf{ex}} = \{fm_1, fm_2, fm_3\}$ with three members: $fm_1 = (G_1, I_{\mathsf{brTPF}})$, $fm_2 = (G_2, I_{\mathsf{TPF}})$, and $fm_3 = (G_3, I_{\mathsf{sparql}})$. The data in these members are the following RDF graphs.

$G_1 = \{(\mathsf{a}, \mathsf{foaf{:}knows}, \mathsf{c})\}$ $\quad$ $G_2 = \{(\mathsf{c}, \mathsf{foaf{:}name}, \texttt{"Lee"}), (\mathsf{d}, \mathsf{foaf{:}name}, \texttt{"Alice"})\}$

$G_3 = \{(\mathsf{a}, \mathsf{foaf{:}knows}, \mathsf{b}), (\mathsf{b}, \mathsf{foaf{:}name}, \texttt{"Peter"})\}$

Informally, the result of a SPARQL query over such a federation should be the same as if the query was executed over the union of all the RDF data available in all the federation members. Formally, this query semantics is defined as follows.

Definition 3. The **evaluation** of a SPARQL graph pattern P over a federation F, denoted by $[\![P]\!]_F$, is a set of solution mappings that is defined as: $[\![P]\!]_F := [\![P]\!]_{G_{\mathrm{union}}}$ where $G_{\mathrm{union}} = \bigcup_{(G,I) \in F} G$ (and $[\![P]\!]_{G_{\mathrm{union}}}$ is as in Section 2).

Example 5. Consider a BGP $B_{\mathsf{ex}} = \{tp_1, tp_2\}$ with the two triple patterns $tp_1 = (?x, \mathsf{foaf{:}knows}, ?y)$ and $tp_2 = (?y, \mathsf{foaf{:}name}, ?z)$. When evaluating B_{ex} over our example federation F_{ex} in Example 4, we obtain $[\![B_{\mathsf{ex}}]\!]_{F_{\mathsf{ex}}} = \{\mu_1, \mu_2\}$ with

$$\mu_1 = \{?x \to \mathsf{a}, ?y \to \mathsf{c}, ?z \to \texttt{"Lee"}\}, \text{ and}$$
$$\mu_2 = \{?x \to \mathsf{a}, ?y \to \mathsf{b}, ?z \to \texttt{"Peter"}\}.$$

For any interface $I = (L_{\mathrm{req}}, \varrho)$, we say that I *supports triple pattern requests* if we can write every triple pattern tp as a request ρ in L_{req} such that for every RDF graph G we have that $\varrho(\rho, G) = [\![tp]\!]_G$. Similarly, I *supports BGP requests* if every BGP B can be written as a request ρ in L_{req} such that for every RDF graph G we have that $\varrho(\rho, G) = [\![B]\!]_G$. Then, out of the three aforementioned interfaces (cf. Examples 1–3), only the SPARQL endpoint interface supports BGP requests, but all three support triple pattern requests.

The notion of support for triple pattern requests (resp. BGP requests) can be carried over to federation members; e.g., if the interface I of a federation member $fm = (G, I)$ supports triple pattern requests, we also say that *fm supports triple pattern requests*.

Finally, we say that a federation is *triple pattern accessible* if all of its members support triple pattern requests. For the complexity results in this paper (cf. Section 5.6) we assume that such federations can be encoded on the tape of a Turing Machine such that all triples that match a given triple pattern can be found in polynomial time.

4 QUERY PLAN LANGUAGE

Now we introduce our language, FedQPL, to describe logical plans for executing queries over heterogeneous federations of RDF data.

A logical plan is a tree of algebraic operators that capture a declarative notion of how their output is related to their input (rather than a concrete algorithm of how the output will be produced, which is the focus of physical plans). Additionally, a logical plan typically also indicates an order in which the operators will be evaluated.

As mentioned in the introduction, such plans are presented only informally in existing work on query engines for federations of RDF data. In contrast to these informal representations, we define both the syntax and the semantics of FedQPL formally. In comparison to the standard SPARQL algebra, the main innovations of FedQPL are that it contains operators to make explicit which federation member is accessed in each part of a query plan and to distinguish different ways of accessing a federation member.

We begin by defining the syntax of FedQPL.

Definition 4. A **FedQPL expression** is an expression φ that can be constructed from the following grammar, in which *req*, *tpAdd*, *bgpAdd*, *join*, *union*, *mj*, *mu*, (, and) are terminal symbols, ρ is an expression in the request language L_{req} of some interface, fm is a federation member, tp is a triple pattern, B is a BGP, and Φ is a nonempty set of FedQPL expressions.

$$\varphi ::= \mathit{req}^{\rho}_{fm} \mid \mathit{tpAdd}^{tp}_{fm}(\varphi) \mid \mathit{bgpAdd}^{B}_{fm}(\varphi) \mid \mathit{join}(\varphi, \varphi) \mid \mathit{union}(\varphi, \varphi) \mid \mathit{mj}\,\Phi \mid \mathit{mu}\,\Phi$$

Before we present the formal semantics of FedQPL expressions, we provide an intuition of the different operators of the language.

4.1 Informal Overview and Intended Use

The first operator, *req*, captures the intention to retrieve the result of a given (sub)query from a given federation member, where the (sub)query is expressed in the request language of the interface provided by the federation member.

Example 6. For BGP $B_{\mathsf{ex}} = \{tp_1, tp_2\}$ in Example 5 we observe that member fm_1 of our example federation F_{ex} (cf. Example 4) can contribute a solution mapping for tp_1, whereas fm_2 can contribute two solution mappings for tp_2. The intention to retrieve these solution mappings from these federation members can be represented in a logical plan by the operators $\mathit{req}^{tp_1}_{fm_1}$ and $\mathit{req}^{tp_2}_{fm_2}$, respectively.

Example 7. We also observe that, by accessing federation member $fm_3 \in F_{\mathsf{ex}}$, we may retrieve a nonempty result for the example

BGP B_{ex} as a whole. The intention to do so can be represented by the operator $req_{fm_3}^{\{tp_1, tp_2\}}$. Notice that such a request with a BGP (rather than with a single triple pattern) is possible only because the interface of fm_3 is the SPARQL endpoint interface I_{sparql} (cf. Example 4) which supports BGP requests.

The second operator, *tpAdd*, captures the intention to access a federation member to obtain solution mappings for a single triple pattern that need to be compatible with (and are to be joined with) solution mappings in a given intermediate query result.

Example 8. Continuing with our example BGP (cf. Example 5) over our example federation F_{ex} (cf. Example 4), we observe that the solution mapping for tp_1 from fm_1 can be joined with only one of the solution mappings for tp_2 from fm_2. To produce the join between the two sets of solution mappings (i.e., between $[\![tp_1]\!]_{G_1}$ and $[\![tp_2]\!]_{G_2}$) we may use the set $[\![tp_1]\!]_{G_1}$ as input to retrieve only those solution mappings for tp_2 from fm_2 that can actually be joined with the solution mapping in $[\![tp_1]\!]_{G_1}$. The plan to do so can be represented by combining a *tpAdd* operator with the *req* operator that retrieves $[\![tp_1]\!]_{G_1}$, which gives us the following FedQPL expression.

$$tpAdd_{fm_2}^{tp_2}\left(req_{fm_1}^{tp_1}\right)$$

While the third operator, *bgpAdd*, represents a BGP-based variation of *tpAdd*, the remaining operators (*join*, *union*, *mj*, and *mu*) lift the standard SPARQL algebra operators join and union into the FedQPL language. In particular, *join* is a binary operator that joins two inputs whereas *mj* represents a multiway variation of a join that can combine an arbitrary number of inputs. In contrast to *tpAdd* and *bgpAdd*, the operators *join* and *mj* capture the intention to obtain the input sets of solution mappings independently and, then, join them only in the query federation engine alone. The operators *union* and *mu* are the union-based counterparts of *join* and *mj*. Our language contains both the binary and the multiway variations of these operators to allow for query plans in which the intention to apply a multiway algorithm can be distinguished explicitly from the intention to use some algorithm designed for the binary case; additionally, the operators *mj* and *mu* can be used during early stages of query planning when the order in which multiple intermediate results will be combined is not decided yet.

Example 9. By executing the operator $req_{fm_3}^{\{tp_1, tp_2\}}$ as discussed in Example 7, we can obtain a solution mapping that is part of the query result for our example BGP B_{ex} (cf. Example 5). Another such part of this result may be obtained based on the FedQPL expression in Example 8. These partial results can then be combined by using the *union* operator as follows.

$$union\Big(\ req_{fm_3}^{\{tp_1, tp_2\}}\ ,\ tpAdd_{fm_2}^{tp_2}\left(req_{fm_1}^{tp_1}\right)\ \Big)$$

To further elaborate on the distinction between the *join* operator (as well as its multiway counterpart *mj*) and the operators *tpAdd* and *bgpAdd* we emphasize that the latter can be used in cases in which the processing power of a federation member can be exploited to join an input set of solution mappings with the result of evaluating a triple pattern (or a BGP) over the data of that federation member. Specific algorithms that can be used as implementations of *tpAdd* (or *bgpAdd*) in such cases are RDF-specific variations of the semijoin [5] and the bind join [11]. Concrete examples of such algorithms can be found in the SPARQL endpoint federation engines FedX [25], SemaGrow [7], and CostFed [22], as well as in the brTPF client [14]. Such algorithms rely on a data access interface in which the given input solution mappings can be captured as part of the requests. However, for less expressive interfaces (such as the TPF interface), the *tpAdd* operator can also be implemented using a variation of an index nested-loops join in which a separate request is created for each input solution mapping [21, 27, 29]. In contrast, a standard (local) nested-loops join—which has also been proposed in the literature on SPARQL federation engines [21]—would be an implementation of the *join* operator. Further examples of join algorithms that have been proposed for such engines and that would be implementations of the *join* operator are a group join [29], a simple hash join [3, 10], a symmetric hash join [3, 22], and a merge join [7].

4.2 Validity

While the various FedQPL operators can be combined arbitrarily as per the grammar in Definition 4, not every operator can be used arbitrarily for every federation member. In contrast, as already indicated in Example 7, depending on their interface, federation members may not be capable to be accessed in the way as required by a particular operator. This observation leads to a notion of validity of FedQPL expressions that we define recursively as follows.

Definition 5. Let F be a federation. A FedQPL expression φ **is valid for** F if it has the following properties:

(1) if φ is of the form req_{fm}^{ρ} where $fm = (G, I)$ with $I = (L_{\text{req}}, \varrho)$, then $fm \in F$ and ρ is an expression in L_{req};

(2) if φ is of the form $tpAdd_{fm}^{tp}(\varphi')$, then $fm \in F$, fm supports triple pattern requests, and φ' is valid;

(3) if φ is of the form $bgpAdd_{fm}^{B}(\varphi')$, then $fm \in F$, fm supports BGP requests, and φ' is valid;

(4) if φ is of the form $join(\varphi_1, \varphi_2)$, then φ_1 and φ_2 are valid;

(5) if φ is of the form $union(\varphi_1, \varphi_2)$, then φ_1 and φ_2 are valid;

(6) if φ is of the form $mj\Phi$, then every $\varphi' \in \Phi$ is valid;

(7) if φ is of the form $mu\Phi$, then every $\varphi' \in \Phi$ is valid.

Example 10. All FedQPL expressions presented in the previous examples are valid for our example federation F_{ex} in Example 4.

Example 11. An expression that is not valid for F_{ex} is $req_{fm_1}^{\{tp_1, tp_2\}}$. The issue with this expression—and with every other expression that contains it as a subexpression—is that it assumes that federation member fm_1 supports BGP requests, which is not the case because the interface of fm_1 is I_{brTPF}. For the same reason, fm_1 cannot be used in the *bgpAdd* operator, which makes expressions such as $bgpAdd_{fm_2}^{\{tp_1, tp_2\}}(\varphi)$ to be invalid as well (for any subexpression φ).

In the remainder of this paper we assume FedQPL expressions that are valid for the federation for which they have been created.

4.3 Semantics

Now we are ready to define a formal semantics of FedQPL. To this end, we introduce a function that defines for each (valid) FedQPL expression, the set of solution mappings that is expected as the result of evaluating the expression.

Definition 6. Let φ be a FedQPL expression that is valid for a federation F. The **solution mappings obtained with** φ, denoted by $\mathsf{sols}(\varphi)$, is a set of solution mappings that is defined recursively:

(1) if φ is of the form req^{ρ}_{fm} where $fm = (G, I)$ with $I = (L_{\mathsf{req}}, \varrho)$, then $\mathsf{sols}(\varphi) := \varrho(\rho, G)$;

(2) if φ is $tpAdd^{tp}_{fm}(\varphi')$ where $fm = (G, I)$ with $I = (L_{\mathsf{req}}, \varrho)$, then $\mathsf{sols}(\varphi) := \mathsf{sols}(\varphi') \bowtie \varrho(tp, G)$;

(3) if φ is $bgpAdd^{B}_{fm}(\varphi')$ where $fm = (G, I)$ with $I = (L_{\mathsf{req}}, \varrho)$, then $\mathsf{sols}(\varphi) := \mathsf{sols}(\varphi') \bowtie \varrho(B, G)$;

(4) if φ is $join(\varphi_1, \varphi_2)$, then $\mathsf{sols}(\varphi) := \mathsf{sols}(\varphi_1) \bowtie \mathsf{sols}(\varphi_2)$;

(5) if φ is $union(\varphi_1, \varphi_2)$, then $\mathsf{sols}(\varphi) := \mathsf{sols}(\varphi_1) \cup \mathsf{sols}(\varphi_2)$;

(6) if φ is of the form $mj\Phi$ where $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$, then $\mathsf{sols}(\varphi) := \mathsf{sols}(\varphi_1) \bowtie \mathsf{sols}(\varphi_2) \bowtie \ldots \bowtie \mathsf{sols}(\varphi_n)$;

(7) if φ is of the form $mu\Phi$ where $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$, then $\mathsf{sols}(\varphi) := \mathsf{sols}(\varphi_1) \cup \mathsf{sols}(\varphi_2) \cup \ldots \cup \mathsf{sols}(\varphi_n)$.

Example 12. Given our example federation F_{ex} (cf. Example 4), for the FedQPL expressions in Examples 7 and 8 we have that

$$\mathsf{sols}(req^{\{tp_1, tp_2\}}_{fm_3}) = \{\mu_2\} \quad \text{and} \quad \mathsf{sols}(tpAdd^{tp_2}_{fm_2}\big(req^{tp_1}_{fm_1}\big)) = \{\mu_1\},$$

where the solution mappings μ_1 and μ_2 are as given in Example 5. Consequently, for the expression in Example 9 we thus have that

$$\mathsf{sols}(\ union\big(req^{\{tp_1, tp_2\}}_{fm_3}, tpAdd^{tp_2}_{fm_2}\big(req^{tp_1}_{fm_1}\big)\big)\ ) = \{\mu_1, \mu_2\}.$$

4.4 Correctness

Given that FedQPL expressions are meant to represent (logical) query execution plans to produce the result of a given BGP over a given federation, we also introduce a correctness property to indicate whether a FedQPL expression correctly captures a given BGP for a given federation. Informally, a FedQPL expression has this correctness property if the set of solution mappings obtained with the expression is the result expected for the BGP over the federation.

Definition 7. Let B be a BGP and F be a federation. A FedQPL expression φ **is correct for** B **over** F if φ is valid for F and it holds that $\mathsf{sols}(\varphi) = [\![B]\!]_F$.

Example 13. Based on Examples 5 and 12, we can see that the FedQPL expression in Example 9 is correct for our example BGP B_{ex} over our example federation F_{ex}.

This completes the definition of FedQPL. In the remainder of the paper we first focus on a fragment of the language that can be used to capture the output of the query decomposition & source selection step. Thereafter, we show equivalences for FedQPL expressions, which provide a formal foundation for logical query optimization.

5 SOURCE SELECTION AND INITIAL PLANS

An important aspect and one of the first steps of planning the execution of queries over a federation is to identify which federation members have to be contacted for which part of a given query. The key tasks of this step are referred to as *query decomposition* and *source selection* [2, 9, 19, 28]. In this section, we identify a fragment of FedQPL that can be used to capture the output of this step formally. We call the expressions in this fragment *source assignments*. After defining them, we study their expressive power as well as the complexity of finding minimal source assignments.

We emphasize that this source assignments fragment of FedQPL provides a foundation to define query decomposition and source selection approaches formally, and to compare such approaches systematically. Moreover, and perhaps more importantly from a practical perspective, if the output of such an approach is described in the form of a source assignment, it can readily be used as an *initial logical plan* that can then be rewritten and refined during the subsequent query optimization steps.

5.1 Source Assignments

The goal of query decomposition is to split a given BGP into smaller components, called *subqueries*, which may be subsets of the BGP or even individual triple patterns. The goal of source selection is to assign to each such subquery the federation members from which we may retrieve a nonempty result for the subquery. We may capture such an assignment of a federation member to a subquery by the FedQPL operator *req* for which we only have to consider requests in the form of a BGP or a triple pattern. Additionally, it needs to be specified how these individual assignments belong together such that the intermediate results that may be obtained from them can be combined correctly into the complete result of the given BGP. To this end, we may use the operators *mu* (for intermediate results that cover the same subqueries) and *mj* (for intermediate results of different subqueries). We emphasize that we select the multiway versions of union and join on purpose; they do not prescribe any order over their input operators, which captures more accurately the output of the query decomposition and source selection step (deciding on such an order is not part of this step, but of the subsequent query optimization steps). With this, we have all parts of the language fragment needed for source assignments.

Definition 8. A **source assignment** is a FedQPL expression that uses only the operators *req*, *mj* and *mu*, and for each subexpression of the form req^{ρ}_{fm} it holds that ρ is a triple pattern or a BGP.

Example 14. From the running example we recall that members fm_1 and fm_3 of our example federation F_{ex} can contribute matching triples for tp_1 of the example BGP B_{ex} (cf. Example 5), whereas tp_2 can be matched only in the data of fm_2 and fm_3. Hence, we may use the following source assignment a_{ex} for B_{ex} over F_{ex}.

$$mj\Big\{\ mu\{req^{tp_1}_{fm_1}, req^{tp_1}_{fm_3}\}\ ,\ mu\{req^{tp_2}_{fm_2}, req^{tp_2}_{fm_3}\}\ \Big\}$$

However, the matching triple for tp_1 in fm_1 can be combined only with the triple for tp_2 in fm_2 and, similarly, the triple for tp_1 in fm_3 can be combined only with the triple for tp_2 in fm_3. Hence, a more sophisticated query decomposition & source selection approach may prune one access to fm_3 by combining tp_1 and tp_2 for a single

access to fm_3, which gives us the following source assignment a'_{ex}.

$$mu\Big\{\ mj\{req_{fm_1}^{tp_1}, req_{fm_2}^{tp_2}\}\ ,\ req_{fm_3}^{\{tp_1,tp_2\}}\ \Big\}$$

It can be easily verified that both of these source assignments, a_{ex} and a'_{ex}, are correct for B_{ex} over F_{ex}.

5.2 Exhaustive Source Assignments

In the previous example (Example 14) we assume to have detailed knowledge of the data available at all the federation members. In practice, however, this knowledge may be much more limited and is captured in some form of pre-populated data catalog with metadata about the federation members [2, 9, 19]. The particular types of metadata vary for each source selection approach that relies on such a catalog. On the other hand, there are also source selection approaches that do not use a pre-populated data catalog at all but, instead, aim to obtain some information about the data of the federation members during the source selection process itself [1, 25, 28].

A straightforward approach that does not require any such information is to create a source assignment that requests every triple pattern of the BGP separately at each federation member. Then, the results of these requests can be unioned per triple pattern and, finally, all the triple pattern specific unions can be joined. We call a source assignment that captures this approach *exhaustive*.

Definition 9. Let $F = \{fm_1, fm_2, \ldots, fm_m\}$ be a federation that is triple pattern accessible, and let $B = \{tp_1, tp_2, \ldots, tp_n\}$ be a BGP. The **exhaustive source assignment** for B over F is the following source assignment.

$$mj\Big\{\ mu\{req_{fm_1}^{tp_1}, \ldots, req_{fm_m}^{tp_1}\}\ , \ldots,\ mu\{req_{fm_1}^{tp_n}, \ldots, req_{fm_m}^{tp_n}\}\ \Big\}$$

The following result follows readily from Definitions 3, 6, 7, 9.

Proposition 1. Let B be a BGP and F be a federation that is triple pattern accessible. The exhaustive source assignment for B over F is correct for B over F.

Corollary 1. By Proposition 1, it follows trivially that for every BGP B and every triple pattern accessible federation F, there exists a source assignment that is correct for B over F.

5.3 Cost and Minimality

While exhaustive source assignments are correct, for many cases there are other source assignments that are also correct but have a smaller number of *req* operators. Smaller numbers are desirable from a performance perspective because each such operator represents the intention to access a given federation member regarding a particular subquery. Hence, we may use this number as a simple cost function to compare source assignments and, ultimately, to compare different query decomposition & source selection approaches.

Definition 10. The **source access cost** (**sa-cost**) of a source assignment a, denoted by sa-cost(a), is the number of subexpressions of the form req_{fm}^{ρ} that are contained (recursively) within a.

Notice that, for the exhaustive source assignment a for a BGP B over a triple pattern accessible federation F: sa-cost(a) = $|B| \cdot |F|$. For other source assignments, the sa-cost may be smaller.

Example 15. The exhaustive source assignment for our example BGP B_{ex} (cf. Example 5) over federation F_{ex} (cf. Example 4) would have an sa-cost of 6. In contrast, the sa-cost of the source assignments a_{ex} and a'_{ex} in Example 14 is 4 and 3, respectively.

Based on our notion of sa-cost, we can also introduce a notion of minimality for source assignments.

Definition 11. Let B be a BGP and F be a federation that is triple pattern accessible. A source assignment a that is correct for B over F is **minimal** for B over F if there is no other source assignment a' such that a' is correct for B over F and sa-cost(a) > sa-cost(a').

Example 16. By comparing the sa-costs of a_{ex} and a'_{ex} in Example 15, we see that a_{ex} is *not* minimal for B_{ex} over F_{ex}. Moreover, it is not difficult to check that a'_{ex} is minimal for B_{ex} over F_{ex}.

Finding minimal source assignments is a problem that resembles the typical optimization problem that existing query decomposition & source selection approaches aim to solve for homogeneous federations (e.g., [1, 3, 7, 22, 25, 28]). Before we study the complexity of this problem for our more general case of heterogeneous federations, we demonstrate the suitability of source assignments as a formal foundation to capture the output of these existing approaches, which also allows us to show limitations of these approaches.

5.4 Application to Existing Approaches

This section illustrates how our notion of source assignments can be applied to describe the output of several existing source selection approaches. To this end, we consider the query LS6 from the FedBench benchmark [23], which consists of a BGP with five triple patterns, $B_{\text{LS6}} = \{tp_1, tp_2, tp_3, tp_4, tp_5\}$ (cf. Listing 1).

```
SELECT ?drug ?title WHERE {
  ?drug db:drugCategory dbc:micronutrient .   # tp1
  ?drug db:casRegistryNumber ?id .            # tp2
  ?keggDrug rdf:type kegg:Drug .              # tp3
  ?keggDrug bio2rdf:xRef ?id .                # tp4
  ?keggDrug purl:title ?title }               # tp5
```

Listing 1: FedBench query LS6 (prefix declarations omitted)

The FedBench benchmark specifies a federation consisting of several members that all provide a SPARQL endpoint interface, where only some of these members are potentially relevant for query LS6. Hence, in terms of our data model, we have a federation F_{FedBench} that, among others, contains the following four members:

$$fm_{\text{drb}} = (DrugBank, I_{\text{sparql}}) \in F_{\text{FedBench}},$$
$$fm_{\text{kegg}} = (KEGG, I_{\text{sparql}}) \in F_{\text{FedBench}},$$
$$fm_{\text{dbp}} = (DBPedia, I_{\text{sparql}}) \in F_{\text{FedBench}},$$
$$fm_{\text{chebi}} = (ChEBI, I_{\text{sparql}}) \in F_{\text{FedBench}}.$$

FedBench query LS6 is interesting for our purpose because different authors have used it as an example to describe their query decomposition & source selection approaches [7, 22, 25]. Hence, the output of these approaches for this query is well documented; yet, the authors present these outputs only informally within a textual description or in figures. Given our notion of source assignments, we now can provide a precise formal description.

federation engine	source assignment	sa-cost
FedX [25]	$mj\Big\{\ req_{fm_{\text{drb}}}^{\{tp_1,tp_2\}},\ req_{fm_{\text{kegg}}}^{\{tp_3,tp_4\}},\ mu\{req_{fm_{\text{drb}}}^{tp_5},\ req_{fm_{\text{kegg}}}^{tp_5},\ req_{fm_{\text{dbp}}}^{tp_5}\ \}\ \Big\}$	5
SemaGrow [7]	$mj\Big\{\ req_{fm_{\text{drb}}}^{\{tp_1,tp_2\}},\ req_{fm_{\text{kegg}}}^{tp_3},\ mu\{req_{fm_{\text{kegg}}}^{tp_4},\ req_{fm_{\text{chebi}}}^{tp_4}\},\ mu\{req_{fm_{\text{kegg}}}^{tp_5},\ req_{fm_{\text{chebi}}}^{tp_5}\}\ \Big\}$	6
CostFed [22]	$mj\Big\{\ req_{fm_{\text{drb}}}^{\{tp_1,tp_2\}},\ req_{fm_{\text{kegg}}}^{\{tp_3,tp_4,tp_5\}}\ \Big\}$	2

Table 1: Source assignments for FedBench query LS6 by different federation engines.

Hence, based on the informal descriptions provided by the authors, we have reconstructed the respective source assignments that the corresponding SPARQL federation engines would produce for BGP B_{LS6} over the federation F_{FedBench}. Table 1 presents these source assignments. We emphasize that all these source assignments are correct for B_{LS6} over F_{FedBench}. Yet, they are syntactically different and have different sa-costs, as also detailed in the table. The source assignment with the lowest sa-cost is found by CostFed, whereas the source assignment of SemaGrow has the highest sa-cost (in this particular case).

5.5 Expressive Power of Source Assignments

While the source assignments of the different approaches in the previous section differ from one another (cf. Table 1), we observe that they are all of the same general form. A related concept that resembles this specific form of source assignments is Vidal et al.'s notion of a "*SPARQL query decomposition*" [28]. These observations raise the following questions: how does Vidal at el.'s notion compare to our notion of source assignments and, ultimately, *what is the expressive power of the form of source assignments that it resembles*?

To address these questions we define Vidal et al.'s notion in terms of the source assignments fragment of FedQPL. To this end, we first notice that a more restricted version of the grammar is sufficient.

Definition 12. The **joins-over-unions class** of source assignments, denoted by $S_{\bowtie_{(\cup)}}$, consists of every source assignment a that can be constructed from the following grammar, where *mj*, *mu*, and *req* are terminal symbols, ρ is a triple pattern or a BGP, *fm* is a federation member, Φ_u is a nonempty set of source assignments that can be formed by the construction a_u, and Φ_b is a nonempty set of source assignments that can be formed by the construction a_b.

$$\begin{array}{lcll} a & ::= & a_u & | \quad mj\,\Phi_u \\ a_u & ::= & a_b & | \quad mu\,\Phi_b \\ a_b & ::= & req_{fm}^{\rho} & \end{array}$$

In addition to restricting the grammar, we need to introduce further syntactic restrictions to accurately capture Vidal et al.'s concept of a SPARQL query decomposition. Namely, in terms of our language, this concept is limited to source assignments in $S_{\bowtie_{(\cup)}}$ for which it holds that, within any subexpression of the form

$$mu\Big\{req_{fm_1}^{\rho_1},\dots,req_{fm_n}^{\rho_n}\Big\},$$

all $\rho_1,\dots,\rho_n$ are the same triple pattern or BGP; additionally, for source assignments of the form $mj\{a_1,\dots,a_n\}$, there cannot be any triple pattern that occurs in more than one of the subexpressions.

For the following formal definition of these additional restrictions we introduce the recursive function subexprs that maps every source assignment a to a set of all (sub)expressions contained in a,

$$\text{subexprs}(a) := \begin{cases} \{a\} & \text{if } a \text{ is of the form } req_{fm}^{\rho}, \\ \{a\} \cup \bigcup_{1\le i\le n}\text{subexprs}(a_i) & \text{if } a \text{ is of the form } mu\{a_1,\dots,a_n\}, \\ \{a\} \cup \bigcup_{1\le i\le n}\text{subexprs}(a_i) & \text{if } a \text{ is of the form } mj\{a_1,\dots,a_n\}. \end{cases}$$

Now we are ready to define the *restricted* joins-over-unions class that captures Vidal et al.'s concept of a "SPARQL query decomposition" accurately.

Definition 13. The **restricted joins-over-unions class** of source assignments, denoted by $S^*_{\bowtie_{(\cup)}}$, consists of every source assignment a that is in $S_{\bowtie_{(\cup)}}$ and that has the following two properties.

(1) for every expression of the form $mu\Big\{req_{fm_1}^{\rho_1},\dots,req_{fm_n}^{\rho_n}\Big\}$ that is in subexprs(a), it holds that $\rho_i = \rho_j$ for all $i,j \in \{1,\dots,n\}$;

(2) if a is of the form $mj\{a_1,\dots,a_n\}$, then $\text{tps}(a_i) \cap \text{tps}(a_j) = \emptyset$ for all $i,j \in \{1,\dots,n\}$, where $\text{tps}(a_k)$ denotes to the set of all the triple patterns mentioned in subexpression a_k for all $k \in \{1,\dots,n\}$, i.e.,

$$\begin{aligned} \text{tps}(a_k) := \{tp \in \rho \mid a' \in \text{subexprs}(a_k) \text{ such that } a' \text{ is of} \\ \text{the form } req_{fm}^{\rho} \text{ where } \rho \text{ is a BGP }\} \\ \cup\ \{\rho \mid a' \in \text{subexprs}(a_k) \text{ such that } a' \text{ is of the} \\ \text{form } req_{fm}^{\rho} \text{ where } \rho \text{ is a triple pattern }\}. \end{aligned}$$

Notice that all exhaustive source assignments (cf. Definition 9) are in the class $S^*_{\bowtie_{(\cup)}}$, and so are the source assignments of Table 1. More generally, to the best of our knowledge, this class encompasses all types of source assignments that the query decomposition & source selection approaches proposed in the literature can produce. A natural question at this point is: *does their restriction to consider only source assignments in $S^*_{\bowtie_{(\cup)}}$ present an actual limitation in the sense that these approaches are inherently unable to find a minimal source assignment in specific cases*?

The following result shows that this is indeed the case.

Proposition 2. There exists a BGP B and a triple pattern accessible federation F such that all source assignments that are both correct and minimal for B over F are not in $S^*_{\bowtie_{(\cup)}}$.

We prove Proposition 2 by showing the following claim, which is even stronger than the claim in Proposition 2.

Lemma 1. There exists a BGP B, a triple pattern accessible federation F, and a source assignment a that is correct for B over F, such that sa-cost(a) < sa-cost(a') for every source assignment a' that is in $S^*_{\bowtie_{(\cup)}}$ and that is also correct for B over F.

Sketch. Lemma 1 can be shown based on our example source assignment a'_{ex} (Example 14) which is not in $S^*_{\bowtie_{(\cup)}}$ and has an sa-cost that is smaller than the sa-cost of any relevant source assignment in $S^*_{\bowtie_{(\cup)}}$. (For a detailed discussion of this and the other proofs in this section, which we have only sketched due to space limitations, refer to the appendix on page 11.) □

As a final remark, we emphasize that the same limitation exists even if we consider the less restricted class $S_{\bowtie_{(\cup)}}$ (instead of $S^*_{\bowtie_{(\cup)}}$).

Lemma 2. There exists a BGP B, a triple pattern accessible federation F, and a source assignment a that is correct for B over F, such that sa-cost(a) < sa-cost(a') for every source assignment a' that is in $S_{\bowtie_{(\cup)}}$ and that is also correct for B over F.

Sketch. The proof of Lemma 1 also proves Lemma 2 because the source assignment a'_{ex} is also not in $S_{\bowtie_{(\cup)}}$. □

5.6 Complexity of Source Selection

For a version of the source selection problem that is defined based on their notion of a "SPARQL query decomposition," Vidal et al. show that this problem is NP-hard [28]. Since we know from the previous section that this notion does not provide the full expressive power of our notion of source assignments and, furthermore, Vidal et al.'s work focuses only on homogeneous federations (of SPARQL endpoints), it is interesting to study the complexity of source selection for our more general case. To this end, we formulate the following decision problem.

Definition 14. Given a BGP B, a triple pattern accessible federation F, and a positive integer c, the **source selection problem** is to decide whether there exists a source assignment a such that a is correct for B over F and sa-cost$(a) \leq c$.

Unfortunately, it can be shown that this problem is also NP-hard.

Theorem 1. The source selection problem is NP-hard.

Sketch. We show the NP-hardness by a reduction from the node cover problem (also called vertex cover problem), which is known to be NP-hard [16]. The detailed proof is in the appendix (page 11). □

The following theorem also gives an upper bound for the complexity of the source selection problem in Definition 14 (note that Vidal et al. do not show such a result for their version of the source selection problem [28]).

Theorem 2. The source selection problem is in Σ^{P}_2.

Sketch. For this proof we assume a nondeterministic Turing machine that guesses a source assignment a and, then, checks that a is correct for B over F and that sa-cost$(a) \leq c$. While sa-cost$(a) \leq c$ can be checked in polynomial time by scanning a, for the correctness check, the machine uses an NP oracle (see the appendix). □

6 EQUIVALENCES

While initial logical plans resulting from the query decomposition & source selection step can produce the correct query results, they may not be efficient in many cases. A query optimizer can convert such plans into more efficient ones by systematically replacing subexpressions by other subexpressions that are semantically equivalent (i.e., that are guaranteed to produce the same result). This section provides a solid formal foundation for such optimizations by showing a comprehensive set of such equivalences for FedQPL.

Before we begin, we need to define the notion of *semantic equivalence* for FedQPL expressions which is federation dependent.

Definition 15. Let F be a federation. Two FedQPL expressions φ and φ' that are valid for F are **semantically equivalent** for F, denoted by $\varphi \overset{F}{\equiv} \varphi'$, if it holds that sols$(\varphi)$ = sols(φ').

Now, the first equivalences cover expressions that focus on a single federation member with an interface that supports *triple patterns requests* (recall from Section 3 that, among others, this includes the SPARQL endpoint interface, the TPF interface, and the brTPF interface). These equivalences follow from Definition 6.

Proposition 3. Let $fm = (G, I)$ be a member in a federation F such that $I = (L_{\text{req}}, \varrho)$ is some interface that supports triple pattern requests; let tp be a triple pattern, and φ and φ' be FedQPL expressions that are valid for F. It holds that:

(1) $join\Big(req^{tp}_{fm}\,,\,\varphi\Big) \overset{F}{\equiv} tpAdd^{tp}_{fm}(\varphi)$;

(2) $join\Big(req^{tp}_{fm}\,,\,join(\varphi,\varphi')\Big) \overset{F}{\equiv} join\Big(tpAdd^{tp}_{fm}(\varphi)\,,\,\varphi'\Big)$.

Sketch. Since all equivalences in this section can be shown in a similar manner by applying Definition 6, we illustrate the proof only for Equivalence (1) and leave the rest as an exercise for the reader. To prove Equivalence (1), we have to show that sols$(join(req^{tp}_{fm}, \varphi))$ = sols$(tpAdd^{tp}_{fm}(\varphi))$, which we do based on Definition 6 as follows (where we can assume that $tp \in L_{\text{req}}$):

$$\begin{aligned}
\text{sols}(join(req^{tp}_{fm}, \varphi)) &= \text{sols}(req^{tp}_{fm}) \bowtie \text{sols}(\varphi) && \text{by Def. 6, Case (4)}\\
&= \varrho(\rho, G) \bowtie \text{sols}(\varphi) && \text{by Def. 6, Case (1)}\\
&= \text{sols}(\varphi) \bowtie \varrho(\rho, G) && \text{by the commutativity of } \bowtie \text{ [20]}\\
&= \text{sols}(tpAdd^{tp}_{fm}(\varphi)) && \text{by Def. 6, Case (2).}
\end{aligned}$$ □

Example 17. By applying Equivalence (1) (cf. Proposition 3), we may rewrite the FedQPL expression in Example 8 into the following expression which is semantically equivalent for F_{ex}.

$$join(req^{tp_2}_{fm_2}, req^{tp_1}_{fm_1})$$

The following equivalences focus on expressions with a federation member that supports *BGP requests*. These equivalences follow from the definition of BGPs (cf. Section 2), in combination with Definition 6. Notice that the first two of these equivalences are the BGP-specific counterparts of the equivalences in Proposition 3.

Proposition 4. Let $fm = (G, I)$ be a member in a federation F such that I is some interface that supports BGP requests; let B, B_1, B_2, and B' be BGPs such that $B' = B_1 \cup B_2$, and φ and φ' be FedQPL expressions that are valid for F. It holds that:

(3) $join\Big(req^{B}_{fm}\,,\,\varphi\Big) \overset{F}{\equiv} bgpAdd^{B}_{fm}(\varphi)$;

(4) $join\Big(req^{B}_{fm}\,,\,join(\varphi,\varphi')\Big) \overset{F}{\equiv} join\Big(bgpAdd^{B}_{fm}(\varphi)\,,\,\varphi'\Big)$;

(5) $join\Big(req^{B_1}_{fm}\,,\,req^{B_2}_{fm}\Big) \overset{F}{\equiv} req^{B'}_{fm}$;

(6) $bgpAdd_{fm}^{B_1}\left(req_{fm}^{B_2}\right) \overset{F}{\equiv} req_{fm}^{B'}$;

(7) $bgpAdd_{fm}^{B_1}\left(bgpAdd_{fm}^{B_2}(\varphi)\right) \overset{F}{\equiv} bgpAdd_{fm}^{B'}(\varphi)$.

The following equivalences focus on expressions with a federation member whose interface supports *both, triple patterns requests and BGP requests*. These equivalences follow from the definition of BGPs, in combination with Definition 6.

Proposition 5. Let $fm = (G, I)$ be a member in a federation F such that I is some interface that supports triple pattern requests as well as BGP requests; let tp be a triple pattern, $B = \{tp_1, tp_2, \ldots, tp_n\}$ be a BGP, and φ be a FedQPL expression that is valid for F. It holds that:

(8) $req_{fm}^{tp} \overset{F}{\equiv} req_{fm}^{B'}$, where $B' = \{tp\}$;

(9) $req_{fm}^{B} \overset{F}{\equiv} join\Big(join\Big(\ldots join\Big(req_{fm}^{tp_1}, req_{fm}^{tp_2}\Big),\ \ldots\Big), req_{fm}^{tp_n}\Big)$;

(10) $req_{fm}^{B} \overset{F}{\equiv} tpAdd_{fm}^{tp_n}(\ldots(tpAdd_{fm}^{tp_2}(req_{fm}^{tp_1}))\ldots)$;

(11) $bgpAdd_{fm}^{B}(\varphi) \overset{F}{\equiv} tpAdd_{fm}^{tp_n}(\ldots(tpAdd_{fm}^{tp_2}(tpAdd_{fm}^{tp_1}(\varphi)))\ldots)$;

(12) $bgpAdd_{fm}^{B}(req_{fm}^{tp}) \overset{F}{\equiv} req_{fm}^{B'}$, where $B' = B \cup \{tp\}$;

(13) $tpAdd_{fm}^{tp}(req_{fm}^{B}) \overset{F}{\equiv} req_{fm}^{B'}$, where $B' = B \cup \{tp\}$;

(14) $tpAdd_{fm}^{tp}\left(bgpAdd_{fm}^{B}(\varphi)\right) \overset{F}{\equiv} bgpAdd_{fm}^{B'}(\varphi)$, where $B' = B \cup \{tp\}$.

The following equivalences focus on expressions with a federation member that provides the *brTPF interface* (cf. Example 3). Consequently, these equivalences follow from the definition of that interface, in combination with Definition 6.

Proposition 6. Let $fm = (G, I_{\text{brTPF}})$ be a member in a federation F such that I_{brTPF} is the brTPF interface; let tp be a triple pattern, Ω be a set of solution mappings, and φ be a FedQPL expression that is valid for F. It holds that:

(15) $join\Big(req_{fm}^{tp}, \varphi\Big) \overset{F}{\equiv} req_{fm}^{(tp,\Omega)}$, where $\Omega = \text{sols}(\varphi)$;

(16) $tpAdd_{fm}^{tp}(\varphi) \overset{F}{\equiv} req_{fm}^{(tp,\Omega)}$, where $\Omega = \text{sols}(\varphi)$.

The following equivalences focus on expressions with federation members that provide the *SPARQL endpoint interface* (cf. Example 1). They follow from the definition of SPARQL graph patterns [12, 20], in combination with Definition 6.

Proposition 7. Let $fm = (G, I_{\text{sparql}})$ be a member in a federation F where I_{sparql} is the SPARQL endpoint interface; let tp be a triple pattern, B be a BGP, and P, P_1 and P_2 be graph patterns. It holds that:

(17) $req_{fm}^{P_1} \overset{F}{\equiv} req_{fm}^{P_2}$ if P_1 and P_2 are semantically equivalent [20];

(18) $union\Big(req_{fm}^{P_1}, req_{fm}^{P_2}\Big) \overset{F}{\equiv} req_{fm}^{(P_1 \text{ UNION } P_2)}$;

(19) $join\Big(req_{fm}^{P_1}, req_{fm}^{P_2}\Big) \overset{F}{\equiv} req_{fm}^{(P_1 \text{ AND } P_2)}$;

(20) $tpAdd_{fm}^{tp}(req_{fm}^{P}) \overset{F}{\equiv} req_{fm}^{(P \text{ AND } tp)}$;

(21) $bgpAdd_{fm}^{B}(req_{fm}^{P}) \overset{F}{\equiv} req_{fm}^{(P \text{ AND } B)}$;

The following equivalences cover expressions with *multiple federation members*. These equivalences follow from Definition 6.

Proposition 8. Let $fm_1 = (G, I_1)$, $fm_2 = (G, I_2)$, $fm_3 = (G, I_3)$, and $fm_4 = (G, I_4)$ be members in a federation F (not necessarily different ones) such that interfaces I_1 and I_2 support triple pattern requests and interfaces I_3 and I_4 support BGP requests. Let tp, tp_1 and tp_2 be triple patterns, B, B_1 and B_2 be BGPs, and φ be a FedQPL expression that is valid for F. It holds that:

(22) $tpAdd_{fm_1}^{tp_1}\left(tpAdd_{fm_2}^{tp_2}(\varphi)\right) \overset{F}{\equiv} tpAdd_{fm_2}^{tp_2}\left(tpAdd_{fm_1}^{tp_1}(\varphi)\right)$;

(23) $tpAdd_{fm_1}^{tp}\left(bgpAdd_{fm_3}^{B}(\varphi)\right) \overset{F}{\equiv} bgpAdd_{fm_3}^{B}\left(tpAdd_{fm_1}^{tp}(\varphi)\right)$;

(24) $bgpAdd_{fm_3}^{B_1}\left(bgpAdd_{fm_4}^{B_2}(\varphi)\right) \overset{F}{\equiv} bgpAdd_{fm_4}^{B_2}\left(bgpAdd_{fm_3}^{B_1}(\varphi)\right)$.

The following equivalences are independent of interface types and focus on the relationships between the two multiway operators (*mj* and *mu*) and their respective binary counterparts.

Proposition 9. Let F be a federation, and let Φ, Φ_{J}, and Φ_{U} be sets of FedQPL expressions that are valid for F such that $|\Phi| > 1$, there exists $join(\varphi_1, \varphi_2) \in \Phi_{\text{J}}$ and $union(\varphi_1, \varphi_2) \in \Phi_{\text{U}}$. It holds that:

(25) $mj\,\Phi \overset{F}{\equiv} join(mj\,\Phi', \varphi)$, where $\varphi \in \Phi$ and $\Phi' = \Phi - \{\varphi\}$;

(26) $mu\,\Phi \overset{F}{\equiv} union(mu\,\Phi', \varphi)$, where $\varphi \in \Phi$ and $\Phi' = \Phi - \{\varphi\}$;

(27) $mj\,\Phi_{\text{J}} \overset{F}{\equiv} mj\,\Phi'$, where $\Phi' = (\Phi_{\text{J}} - \{join(\varphi_1, \varphi_2)\}) \cup \{\varphi_1, \varphi_2\}$;

(28) $mu\,\Phi_{\text{U}} \overset{F}{\equiv} mu\,\Phi'$, where $\Phi' = (\Phi_{\text{U}} - \{union(\varphi_1, \varphi_2)\}) \cup \{\varphi_1, \varphi_2\}$.

The following equivalences are also independent of interface types and focus only on the two multiway operators.

Proposition 10. Let F be a federation; let φ be a FedQPL expression that is valid for F, and let Φ_{J} and Φ_{U} be sets of FedQPL expressions that are valid for F such that there exists $mj\,\Phi'_{\text{J}} \in \Phi_{\text{J}}$ and $mu\,\Phi'_{\text{U}} \in \Phi_{\text{U}}$. It holds that:

(29) $mj\,\Phi_{\text{J}} \overset{F}{\equiv} mj\,\Phi''_{\text{J}}$, where $\Phi''_{\text{J}} = (\Phi_{\text{J}} - \{mj\,\Phi'_{\text{J}}\}) \cup \Phi'_{\text{J}}$;

(30) $mu\,\Phi_{\text{U}} \overset{F}{\equiv} mu\,\Phi''_{\text{U}}$, where $\Phi''_{\text{U}} = (\Phi_{\text{U}} - \{mu\,\Phi'_{\text{U}}\}) \cup \Phi'_{\text{U}}$;

(31) $mu\{\varphi\} \overset{F}{\equiv} \varphi$;

(32) $mj\{\varphi\} \overset{F}{\equiv} \varphi$.

Lastly, the following equivalences are also independent of interface types; they follow from Definition 6 and the corresponding equivalences for the SPARQL algebra [20, 24].

Proposition 11. Let F be a federation and let φ_1, φ_2, and φ_3 be FedQPL expressions that are valid for F. It holds that:

(33) $join(\varphi_1, \varphi_2) \overset{F}{\equiv} join(\varphi_2, \varphi_1)$;

(34) $union(\varphi_1, \varphi_2) \overset{F}{\equiv} union(\varphi_2, \varphi_1)$;

(35) $union(\varphi_1, \varphi_1) \overset{F}{\equiv} \varphi_1$;

(36) $join\Big(\varphi_1, join(\varphi_2, \varphi_3)\Big) \overset{F}{\equiv} join\Big(join(\varphi_1, \varphi_2), \varphi_3\Big)$;

(37) $union\Big(\varphi_1, union(\varphi_2, \varphi_3)\Big) \overset{F}{\equiv} union\Big(union(\varphi_1, \varphi_2), \varphi_3\Big)$;

(38) $join\Big(\varphi_1, union(\varphi_2, \varphi_3)\Big) \overset{F}{\equiv} union\Big(join(\varphi_1, \varphi_2), join(\varphi_1, \varphi_3)\Big)$.

Example 18. By applying Equivalence (33) (cf. Proposition 11), we may swap the two subexpressions in the FedQPL expression in Example 17, which results in the following expression.

$$join(req_{fm_1}^{tp_1}, req_{fm_2}^{tp_2})$$

We may now rewrite this expression into the following one by applying Equivalence (1) (cf. Proposition 3) again.

$$tpAdd_{fm_1}^{tp_1}\left(req_{fm_2}^{tp_2}\right)$$

Observe that the latter expression is, thus, semantically equivalent (for F_{ex}) not only to the previous expression, but also to the expressions in Examples 17 and 8, respectively.

Example 19. The previous example highlights that an alternative to the plan captured by the FedQPL expression $tpAdd_{fm_2}^{tp_2}\left(req_{fm_1}^{tp_1}\right)$ in Example 8 would be—among others—the plan represented by the expression $tpAdd_{fm_1}^{tp_1}\left(req_{fm_2}^{tp_2}\right)$. Considering that the interface of federation member fm_1 is a brTPF interface whereas fm_2 has a TPF interface (cf. Example 4), we may prefer the latter plan because it provides for a greater number of algorithms to choose from during physical query optimization (e.g., implementing *tpAdd* over brTPF may be done by using a bind join algorithm [14], which is impossible with TPF). On the other hand, a query optimizer may estimate a significantly smaller result size for $req_{fm_1}^{tp_1}$ than for $req_{fm_2}^{tp_2}$, which could justify choosing the plan of Example 8.

7 CONCLUDING REMARKS

In this paper, in addition to proving several important results regarding the query plan language that we propose, we have demonstrated initial ideas for applying this language.

For instance, we have shown that our source assignments can represent the output of existing query decomposition & source selection approaches (cf. Section 5.4), and that these approaches are inherently limited (cf. Proposition 2). Hence, there is still an opportunity for future work on better source selection approaches even in the context of homogeneous federations of SPARQL endpoints.

Moreover, we have not only demonstrated that the language can be used to represent logical query plans (cf. Examples 6–9 and 13), but also that it is suitable both as a basis for logical query optimization (cf. Examples 17–18) and as a starting point for physical query optimization (cf. Example 19). As future work regarding FedQPL, we are planning to extend the language with additional operators that can be used to represent query plans for more expressive fragments of SPARQL, and we aim to also provide a multiset semantics.

However, the ultimate next step is to develop effective optimization approaches for queries over federations with heterogeneous interfaces. FedQPL provides a formal foundation for such work.

ACKNOWLEDGMENTS

Sijin Cheng's work was funded by CUGS (the National Graduate School in Computer Science, Sweden). Olaf Hartig's work was funded in equal parts by the Swedish Research Council (Vetenskapsrådet, project reg. no. 2019-05655) and by the CENIIT program at Linköping University (project no. 17.05).

REFERENCES

[1] Ibrahim Abdelaziz, Essam Mansour, Mourad Ouzzani, Ashraf Aboulnaga, and Panos Kalnis. 2017. Lusail: A System for Querying Linked Data at Scale. *Proceedings of the VLDB Endowment (PVLDB)* 11, 4 (2017).

[2] Maribel Acosta, Olaf Hartig, and Juan F. Sequeda. 2019. Federated RDF Query Processing. In *Encyclopedia of Big Data Technologies*. Springer.

[3] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. 2011. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In *Proc. of the 11th International Semantic Web Conference (ISWC)*.

[4] Amr Azzam, Javier D. Fernández, Maribel Acosta, Martin Beno, and Axel Polleres. 2020. SMART-KG: Hybrid Shipping for SPARQL Querying on the Web. In *Proceedings of The Web Conference (WWW)*.

[5] Philip A. Bernstein, Nathan Goodman, Eugene Wong, Christopher L. Reeve, and James B. Rothnie Jr. 1981. Query Processing in a System for Distributed Databases (SDD-1). *ACM Trans. Database Syst.* 6, 4 (1981).

[6] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. 2013. SPARQL Web-Querying Infrastructure: Ready for Action?. In *Proceedings of the 12th International Semantic Web Conference (ISWC)*.

[7] Angelos Charalambidis, Antonis Troumpoukis, and Stasinos Konstantopoulos. 2015. SemaGrow: Optimizing Federated SPARQL Queries. In *Proc. of the 11th International Conference on Semantic Systems (SEMANTICS)*.

[8] Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark, and Elias Torres. 2013. SPARQL 1.1 Protocol. W3C Recommendation.

[9] Olaf Görlitz and Steffen Staab. 2011. Federated Data Management and Query Optimization for Linked Open Data. In *New Directions in Web Data Management 1*.

[10] Olaf Görlitz and Steffen Staab. 2011. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *Proc. of the 2nd International Workshop on Consuming Linked Data (COLD)*.

[11] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. 1997. Optimizing Queries Across Diverse Data Sources. In *Proc. of the 23rd International Conference on Very Large Data Bases (VLDB)*.

[12] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. 2013. SPARQL 1.1 Query Language. W3C Recommendation, Online at `http://www.w3.org/TR/sparql11-query/`.

[13] Olaf Hartig, Christian Bizer, and Johann Christoph Freytag. 2009. Executing SPARQL Queries over the Web of Linked Data. In *Proc. of the 8th International Semantic Web Conf. (ISWC)*.

[14] Olaf Hartig and Carlos Buil-Aranda. 2016. Bindings-Restricted Triple Pattern Fragments. In *15th Ontologies, Databases, and Applications of Semantics (ODBASE)*.

[15] Olaf Hartig, Ian Letter, and Jorge Pérez. 2017. A Formal Framework for Comparing Linked Data Fragments. In *Proc. of the 16th Int. Semantic Web Conf. (ISWC)*.

[16] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*.

[17] Günter Ladwig and Thanh Tran. 2010. Linked Data Query Processing Strategies. In *Proceedings of the 9th International Semantic Web Conference (ISWC)*.

[18] Thomas Minier, Hala Skaf-Molli, and Pascal Molli. 2019. SaGe: Web Preemption for Public SPARQL Query Services. In *Proceedings of the Web Conference (WWW)*.

[19] Damla Oguz, Belgin Ergenc, Shaoyi Yin, Oguz Dikenelli, and Abdelkader Hameurlain. 2015. Federated Query Processing on Linked Data: A Qualitative Survey and Open Challenges. *Knowledge Eng. Review* 30, 5 (2015).

[20] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and Complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3 (2009).

[21] Bastian Quilitz and Ulf Leser. 2008. Querying Distributed RDF Data Sources with SPARQL. In *Proceedings of the 5th European Semantic Web Conference (ESWC)*.

[22] Muhammad Saleem, Alexander Potocki, Tommaso Soru, Olaf Hartig, and Axel-Cyrille Ngonga Ngomo. 2018. CostFed: Cost-Based Query Optimization for SPARQL Endpoint Federation. In *Proc. of the 14th Int. Conf. on Semantic Systems*.

[23] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. 2011. FedBench: A Benchmark Suite for Federated Semantic Data Query Processing. In *Proc. of the 10th Int. Semantic Web Conference (ISWC)*.

[24] Michael Schmidt, Michael Meier, and Georg Lausen. 2010. Foundations of SPARQL Query Optimization. In *Proc. of the 13th Int. Conference on Database Theory (ICDT)*.

[25] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. 2011. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *Proceedings of the 10th International Semantic Web Conference (ISWC)*.

[26] Jürgen Umbrich, Katja Hose, Marcel Karnstedt, Andreas Harth, and Axel Polleres. 2011. Data Summaries for Processing Live Queries over Linked Data. *World Wide Web* 14, 5-6 (2011).

[27] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. 2016. Triple Pattern Fragments: A Low-Cost Knowledge Graph Interface for the Web. *Journal of Web Semantics* 37 (2016).

[28] Maria-Esther Vidal, Simón Castillo, Maribel Acosta, Gabriela Montoya, and Guillermo Palma. 2016. On the Selection of SPARQL Endpoints to Efficiently Execute Federated SPARQL Queries. *Trans. Large-Scale Data- and Knowledge-Centered Systems* 25 (2016).

[29] Maria-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amadís Martínez, Javier Sierra, and Axel Polleres. 2010. Efficiently Joining Group Patterns in SPARQL Queries. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC)*.

Appendix

PROOF OF LEMMA 1

Lemma 1. There exists a BGP B, a triple pattern accessible federation F, and a source assignment a that is correct for B over F, such that sa-cost(a) < sa-cost(a') for every source assignment a' that is in $S^*_{\bowtie_{(\cup)}}$ and that is also correct for B over F.

Proof. Recall our example source assignment a'_{ex} for BGP B_{ex} over our example federation F_{ex} (cf. Example 14). F_{ex} is triple pattern accessible (cf. Example 4) and a'_{ex} is correct for B_{ex} over F_{ex} (cf. Example 14). Note also that a'_{ex} is clearly not in $S^*_{\bowtie_{(\cup)}}$. Now, we may enumerate every source assignment a' in $S^*_{\bowtie_{(\cup)}}$ that is valid for B_{ex} over F_{ex} and that has an sa-cost smaller or equivalent to 3, i.e., sa-cost$(a') \leq$ sa-cost(a'_{ex}). For each of these source assignments we will find that it is not correct. □

PROOF OF LEMMA 2

Lemma 2. There exists a BGP B, a triple pattern accessible federation F, and a source assignment a that is correct for B over F, such that sa-cost(a) < sa-cost(a') for every source assignment a' that is in $S_{\bowtie_{(\cup)}}$ and that is also correct for B over F.

Proof. We can use the same proof as for Lemma 1 because the source assignment a'_{ex} as used in that proof is also not in $S_{\bowtie_{(\cup)}}$. □

PROOF OF THEOREM 1

Theorem 1. The source selection problem is NP-hard.

Proof. We show the NP-hardness by a reduction from the node cover problem (also called vertex cover problem), which is known to be NP-hard [16].

The node cover problem is defined as follows: Given a positive integer k and an undirected graph $G = (V, E)$, decide whether there exists a set $V' \subseteq V$ such $|V'| \leq k$ and every edge in E is incident on some vertex in V' (i.e., for every undirected edge $\{u, v\} \in E$ it holds that $u \in V'$ or $v \in V'$).

For our reduction we introduce the following function f that maps every instance of the node cover problem (i.e., every $G = (V, E)$ and k) to an instance of the source selection problem (i.e., a BGP B, a triple pattern accessible federation F, and a positive integer c). Given $G = (V, E)$ and k, function f maps the undirected graph G to a federation F that consists of one member per vertex in V. Hence, we define F such that $|F| = |V|$ and there exists a bijection $f_{\text{memb}} : V \to F$. Then, for every vertex $v \in V$, the corresponding federation member $f_{\text{memb}}(v) = (G_v, I_v)$ is defined as follows: The interface I_v is the TPF interface (cf. Example 2), or any other interface that supports triple pattern requests, and for the RDF graph G_v we have that

$$G_v = \left\{ \Big(\text{uri}(e), \texttt{rdf:type}, \texttt{ex:Edge}\Big) \;\middle|\; \begin{array}{l} e \text{ is an edge in } E \\ \text{that is incident on } v \end{array} \right\},$$

where uri$: E \to \mathcal{U}$ is a bijection that maps every edge in E to a distinct URI.

In addition to mapping the input graph G to the federation F, function f maps the integer k directly to c (i.e., $c = k$), and the BGP B returned by f is the same for every instance of the node cover problem; this BGP consists of a single triple pattern: $B = \{tp\}$ where $tp = \Big(?x, \texttt{rdf:type}, \texttt{ex:Edge}\Big)$. It is not difficult to see that our mapping function f can be computed in polynomial time.

Then, the reduction is based on the following claim: For any possible input $G = (V, E)$ and k, and the corresponding output $(B, F, c) = f(G, k)$, there exists a set $V' \subseteq V$ such $|V'| \leq k$ and every edge in E is incident on some vertex in V' if and only if there exists a source assignment a such that a is correct for B over F and sa-cost$(a) \leq c$. This claim is easily verified based on two observations:

(1) For every edge $e = \{u, v\}$ in E, there are exactly two federation members that contain the RDF triple
$$\Big(\text{uri}(e), \texttt{rdf:type}, \texttt{ex:Edge}\Big),$$
namely, the members created for the vertexes v and u, that is, $f_{\text{memb}}(v)$ and $f_{\text{memb}}(u)$.
(2) For at least one of these two members, say fm, a subexpression of the form req^{ρ}_{fm} must be part of every source assignment that is correct for B over F.

Hence, the node cover problem can be reduced to our source selection problem, and since it is NP-hard [16], the source selection problem must be NP-hard as well. □

PROOF OF THEOREM 2

Before we present the proof of Theorem 2, we show two auxiliary results which we shall then use to prove the theorem.

Lemma 3. Given a BGP B, a triple pattern accessible federation F, and a solution mapping μ, the problem to decide whether $\mu \in [\![B]\!]_F$ can be solved in time $O(|F|^2 + |B| \cdot |F|)$.

Proof. To check whether $\mu \in [\![B]\!]_F$ we actually have to check whether $\mu \in [\![B]\!]_{G_{\text{union}}}$ where $G_{\text{union}} = \bigcup_{(G,I)\in F} G$ (cf. Definition 3). To this end, we first materialize the RDF graph G_{union}, which is possible in time $O(|F|^2)$ (note that the complexity is quadratic because of the need to eliminate duplicates when materializing G_{union}). Thereafter, checking whether $\mu \in [\![B]\!]_{G_{\text{union}}}$ is know as the evaluation problem of SPARQL, which has been shown to be solvable in time $O(|B| \cdot |G_{\text{union}}|)$ for BGPs [20]. Since $|F| = |G_{\text{union}}| + k$ for some constant k, the algorithm has an overall time complexity of $O(|F|^2 + |B| \cdot |F|)$. □

Lemma 4. Given a BGP B, a triple pattern accessible federation F, a source assignment a that is valid for B over F, and a solution mapping μ, the problem of deciding whether $\mu \in \text{sols}(a)$ can be solved in time $O(|a| \cdot |F| + |a| \cdot |\mu|)$.

Proof. We proof the lemma by induction over the structure of a.
Base case: If a is of the form req^{ρ}_{fm}, we can use Algorithm 1 to check whether $\mu \in \text{sols}(a)$. By Definition 8, we know that ρ is a triple pattern or a BGP. W.l.o.g., we assume it is a BGP B'. In the algorithm, we write vars(a) to denote the set of variables mentioned in the BGPs within the source assignment a. Formally, this set is defined recursively as follows: If a is of the form req^{ρ}_{fm} where ρ is a triple pattern or a BGP, then vars(a) = vars(ρ). If a is of the form $mu\{a_1, \ldots, a_n\}$ or of the form $mj\{a_1, \ldots, a_n\}$, then vars$(a) = \bigcup_{i\in\{1,\ldots,n\}}$ vars(a_i).

Algorithm 3 Check whether $\mu \in \text{sols}(a)$ for $a = mj\{a_1, \ldots, a_n\}$

```
1: if dom(μ) ≠ vars(a) then
2:     return false          // μ has to be defined for the variables in a
3: end if
4: for all i ∈ {1, ..., n} do
5:     let μ_i be the restriction of μ to the variables in vars(a_i)
6:     if μ_i ∉ sols(a_i) then
7:         return false
8:     end if
9: end for
10: return true
```

Algorithm 1 Check whether $\mu \in \text{sols}(a)$ for $a = req_{fm}^{B'}$ with $fm = (G, I)$

```
1: if dom(μ) ≠ vars(a) then
2:     return false          // μ has to be defined for the variables in a
3: end if
4: let G' := μ[B']           // substitute all variables in B' according to μ
5: for all t ∈ G' do
6:     if t ∉ G then
7:         return false      // search the data of fm for every triple in G'
8:     end if
9: end for
10: return true
```

Algorithm 2 Check whether $\mu \in \text{sols}(a)$ for $a = mu\{a_1, \ldots, a_n\}$

```
1: for all i ∈ {1, ..., n} do
2:     if μ ∈ sols(a_i) then
3:         return true
4:     end if
5: end for
6: return false
```

The first step of the algorithm (lines 1–3) checks that μ is actually defined for the variables in a, which can be done in time $O(|a| \cdot |\mu|)$. The next step is to replace all variables in B' according to μ (cf. line 4), which results in a set G' of RDF triples. The time complexity of this step is again $O(|a| \cdot |\mu|)$ (we assume here that $|a| = |B'| + k$ for some constant k). Finally, the algorithm checks that every triple in G' exists in the data G of federation member fm (lines 5–9). The actual check for each triple (i.e., line 6) can be done in time $O(|F|)$ (we use that $|G| < |F|$) and, thus, the time complexity of the whole for loop (lines 5–9) is $O(|a| \cdot |F|)$. Therefore, the time complexity of the whole algorithm is $O(|a| \cdot |\mu| + |a| \cdot |F|)$.

Induction step: In the induction step we consider the remaining two cases of a.

Case 1) If a is of the form $mu\{a_1, \ldots, a_n\}$, we can check whether $\mu \in \text{sols}(a)$ by using Algorithm 2. The algorithm tries to find a sub-expression a_i inside a such that we have $\mu \in \text{sols}(a_i)$. By the induction hypothesis, the corresponding check in line 2 can be done in time $O(|a_i| \cdot |\mu| + |a_i| \cdot |F|)$ for every $i \in \{1, \ldots, n\}$. Consequently, for the whole loop (lines 1–5), and thus the whole algorithm, we have a time complexity of $O\Big((|a_1| + \ldots + |a_n|) \cdot |\mu| + (|a_1| + \ldots + |a_n|) \cdot |F|\Big)$, which is $O(|a| \cdot |\mu| + |a| \cdot |F|)$.

Case 2) If a is of the form $mj\{a_1, \ldots, a_n\}$, we can check whether $\mu \in \text{sols}(a)$ by using Algorithm 3. The first step (lines 1–3) is to check that μ is actually defined for the variables in a, which can be done in time $O(|a| \cdot |\mu|)$. Thereafter, the algorithm iterates over the subexpressions a_1 to a_n.

For each such subexpression a_i, the algorithm first takes the restriction of μ to the variables in $\text{vars}(a_i)$, denoted by μ_i (line 5); i.e., μ_i is a solution mapping such that $\text{dom}(\mu_i) = \text{dom}(\mu) \cap \text{vars}(a_i)$ and $\mu_i(?v) = \mu(?v)$ for every variable $?v \in \text{dom}(\mu) \cap \text{vars}(a_i)$. For every a_i, this steps can be done in time $O(|a_i| \cdot |\mu|)$. Next, the algorithm checks whether $\mu_i \notin \text{sols}(a_i)$ (line 6), in which case μ cannot be in $\text{sols}(a)$. By the induction hypothesis, for every $i \in \{1, \ldots, n\}$, this check can be done in time $O(|a_i| \cdot |\mu_i| + |a_i| \cdot |F|)$, which we may generalize to $O(|a_i| \cdot |\mu| + |a_i| \cdot |F|)$ because $|\mu| = |\mu_i| + k_i$ for some constant k_i.

Then, the time complexity of the whole for loop (lines 4–9) is $O\Big((|a_1| + \ldots + |a_n|) \cdot |\mu| + (|a_1| + \ldots + |a_n|) \cdot |F|\Big)$, which is $O(|a| \cdot |\mu| + |a| \cdot |F|)$. When combined with the complexity of the first step (lines 1–3, see above), the time complexity of Algorithm 3 is also $O(|a| \cdot |\mu| + |a| \cdot |F|)$. □

Now we are ready to prove the theorem.

Theorem 2. The source selection problem is in Σ_2^{P}.

Proof. We assume a nondeterministic Turing machine (NTM) that is equipped with the following oracle. For every BGP B, every triple pattern accessible federation F, and every source assignment a, the oracle returns `true` if and only if a is *not* correct for B over F. Then, the NTM decides the source selection problem for any given input B, F, and c as follows: First, the NTM guesses a source assignment a such that the size of a is polynomial in the size of B and F (by Proposition 1 we know that such polynomial-sized source assignments exist and are correct for B over F). Then, the NTM has to check that a is correct for B over F and that $\text{sa-cost}(a) \leq c$. The latter property, i.e., $\text{sa-cost}(a) \leq c$, can be checked in polynomial time by scanning a and counting all subexpressions of the form req_{fm}^{ρ}. To check the correctness of a the NTM uses its oracle (and inverts the response of the oracle; i.e., a is correct for B over F if and only if the oracle returns `false`).

Now, to show that the source selection problem is in Σ_2^{P} it remains to show that checking whether a is not correct for B over F (i.e., the decision problem solved by the oracle) is in NP. To this end, we use the following nondeterministic program: First, we test whether a is valid for B over F, which is a precondition for the correctness (cf. Definition 7) and can be checked in polynomial time if we assume that the encoding of F on the tape of a Turing Machine includes an indication of the type of interface that each member in F has. Next, we guess a solution mapping μ where the intuition is that $\mu \in [\![B]\!]_F$ but $\mu \notin \text{sols}(a)$, which shows that a is not correct for B over F. Hence, the program has to check these two properties of μ, which can be done in polynomial time as we have shown in Lemmas 3 and 4. □