The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

On the Satisfiability Problem of Patterns in SPARQL 1.1

Xiaowang Zhang,[1,5] Jan Van den Bussche,[3] Kewen Wang,[2,4] Zhe Wang[4]
[1]School of Computer Science and Technology, Tianjin University, Tianjin, China
[2]School of Computer Software, Tianjin University, Tianjin, China
[3]Faculty of Sciences, Hasselt University, Hasselt, Belgium
[4]School of Information and Communication Technology, Griffith University, Brisbane, Australia
[5]Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China

Abstract

The pattern satisfiability is a fundamental problem for SPARQL. This paper provides a complete analysis of decidability/undecidability of satisfiability problems for SPARQL 1.1 patterns. A surprising result is the undecidability of satisfiability for SPARQL 1.1 patterns when only AND and MINUS are expressible. Also, it is shown that any fragment of SPARQL 1.1 without expressing both AND and MINUS is decidable. These results provide a guideline for future SPARQL query language design and implementation.

Introduction

The Resource Description Framework (RDF), a popular data model for information on the Web, represents information in the form of directed labeled graphs called RDF graphs. The standard query language for RDF data is SPARQL with its latest version SPARQL 1.1 (Harris and Seaborne, 2013). A fundamental problem for SPARQL is that of *satisfiability* of SPARQL patterns. However, the pattern satisfiability for full SPARQL language is undecidable since SPARQL patterns can emulate relational algebra expressions (Angles et al., 2008; Polleres, 2007; Arenas et al., 2011), and satisfiability for relational algebra is undecidable (Abiteboul et al., 1995). For this reason, it would be interesting to investigate computational complexity of pattern satisfiability for fragments of SPARQL 1.1 so that useful decidable or tractable language fragments are identified.

Originally, SPARQL 1.0 contains four operators AND, UNION, OPT and FILTER (for short, $\mathcal{A}$, $\mathcal{U}$, $\mathcal{O}$ and $\mathcal{F}$, respectively). SPARQL 1.1 extends SPARQL 1.0 by introducing six new operators SELECT (expressing subqueries), MINUS, EXISTS, NOT EXISTS, BIND and VALUES (for short, $\mathcal{S}$, $\mathcal{M}$, $\exists$, $\nexists$, $\mathcal{B}$, and $\mathcal{V}$, respectively). While SPARQL 1.1 has some other important new features such as aggregation and property paths (Arenas et al., 2012; Harris and Seaborne, 2013), they are out of the scope of this work. A fragment formed by some of the above operators is denoted as a sequence of abbreviations of the operators. For instance, $\mathcal{AMO}$ is the fragment containing AND, MINUS and OPT.

The satisfiability problem for SPARQL 1.0 patterns has been investigated and some important results are obtained in (Zhang et al., 2016). They showed that the filter operations play an important role on the satisfiability of a SPARQL 1.0 patterns. Especially, without filter operations, a pattern is always satisfiable except for trivial cases where a literal occurs in the wrong place.

SPARQL 1.1 brings in many new language fragments, which make the pattern satisfiability problem of SPARQL 1.1 more interesting and challenging. However, to our best knowledge, this problem has not been explored by researchers yet. In this work, we present decidability/undecidability results for pattern satisfiability of several important fragments of SPARQL 1.1. Moreover, these results allow us to determine decidability/undecidability of all 640 SPARQL 1.1 fragments. Specifically, our major contributions are summarized in the following:

1. *The problem of deciding whether a pattern is satisfiable in $\mathcal{AM}$ is undecidable*. This result is important as it can be used to identify all undecidable fragments of SPARQL 1.1 patterns. It is shown in (Zhang et al., 2016) that pattern satisfiability of the fragment $\mathcal{AMU}$ is undecidable. Their proof relies on the presence of UNION but it is unclear how to express UNION by only AND and MINUS. Fortunately, our result is proven by reducing the satisfiability of the Downward Algebra, which is undecidable, to the pattern decidability of a fragment $\mathcal{AM}^{\mathrm{nav}}$ of $\mathcal{AM}$ (Tan et al., 2014).

2. *The problem of deciding whether a pattern is satisfiable in $\mathcal{BFMUV}(\exists, \nexists)$ is decidable*. This result actually covers all decidable fragments of SPARQL 1.1 that allow neither AND nor OPT. The decidability of pattern satisfiability in $\mathcal{BFMUV}(\exists, \nexists)$ is proven by reducing the satisfiability to that of the guarded fragment of first-order logic, while the latter is decidable (Andréka et al., 1998).

3. *The problem of deciding whether a pattern is satisfiable in $\mathcal{ABFUV}(\exists)$ is decidable*. This result actually covers all decidable OPT-free fragments of SPARQL 1.1 containing AND. It is proven by reducing the satisfiability of $\mathcal{ABFUV}(\exists)$ patterns to that of equality logic, which is decidable (Andréka et al., 1997).

4. *The problems of deciding whether a pattern is satisfiable in $\mathcal{AOU}$ and $\mathcal{BOUV}$ are decidable*. In fact, their satisfiability can be decided in linear time.

5. *We provide a complete picture of decidability/undecidability of all* 640 *fragments of SPARQL 1.1 patterns*.

SPARQL 1.1: Syntax and Semantics

In this section, we briefly recall some basics of SPARQL 1.1, including its syntax, semantics, and the satisfiability problem. We follow definitions and notations for the core SPARQL formalization in (Pérez et al., 2009).

Syntax of SPARQL 1.1 Patterns

An *RDF triple* is a triple of the form $(s, p, o) \in (I \cup B) \times I \times U$, where I, B, and L are infinite sets of *IRIs* (Internationalized Resource Identifier), *blank nodes* and *literals*, respectively, which are pairwise disjoint. The union $I \cup B \cup L$ is denoted by U, and an element of $I \cup L$ is referred to as a *constant*. An *RDF graph* is a finite set of RDF triples.

Note that blank nodes are not constants.

In SPARQL, a query is defined in terms of *patterns*. Assume that V is an infinite set of *variables*, disjoint from U. Following the convention in SPARQL, a variable starts with a question mark to distinguish them from constants. For instance, $?x$ is a variable.

The formalisation of SPARQL 1.1 we consider contains ten operators AND, UNION, OPT, FILTER, SELECT, MINUS, EXISTS, NOT EXISTS, BIND, and VALUES as well as the standard constraints such as $?x = ?y$, $\mathrm{bound}(?x)$ etc. Formally, *patterns* in SPARQL 1.1 are inductively defined as follows.

- Any triple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a pattern (called a *triple pattern*).
- If P_1 and P_2 are patterns, then so are P_1 UNION P_2, P_1 AND P_2, P_1 OPT P_2, $\mathrm{SELECT}_S(P_1)$, P_1 MINUS P_2, P_1 $\mathrm{BIND}_{?x}\,(c)$ (where $?x$ does not occur in P_1), and $(\mathrm{VALUES}\,\vec{W}\,D)$; where $?x \in V$, $c \in I \cup L$, $\vec{W} \in V^n$ (a list of distinct variables), and $D \subseteq (I \cup L)^n$ (a set of vectors of constants) with arity n.
- If P is a pattern and C is a constraint (defined next), then P FILTER C is a pattern; we call C the *filter condition*. Here, a *constraint* can have one of the eight following forms: $\mathrm{bound}(?x)$, $\neg\mathrm{bound}(?x)$, $?x = ?y$, $?x \neq ?y$, $?x = c$, $?x \neq c$, $\mathrm{EXISTS}(P)$, and $\mathrm{NOT\ EXISTS}(P)$, where $c \in I \cup L$.

By a fragment, we mean a collection of patterns formed by a subset of the ten operators of SPARQL 1.1. As explained, such a fragment will be denoted as a sequence of the curlicue initials of these operators. We assume that the standard constraints in SPARQL 1.0 are always allowed in a fragment of SPARQL 1.1.

In the rest of this paper, we omit SELECT since it can be easily removed by a renaming of variables if only pattern satisfiability is considered (Zhang et al., 2016).

Semantics and Satisfiability of Patterns

The semantics of patterns is defined in terms of sets of so-called *solution mappings*. A solution mapping (simply, mapping) is a total function $\mu : S \to U$ on a finite set S of variables. The domain S of μ is denoted $\mathrm{dom}\mu$.

Given a graph G and a pattern P, the semantics $[\![P]\!]_G$ of P on G is defined by a set of mappings as follows:

- $[\![(u, v, w)]\!]_G := \{\mu : \{u, v, w\} \cap V \to U \mid (\mu(u), \mu(v), \mu(w)) \in G\}$. Here for a mapping μ and a constant $c \in I \cup L$, we agree that $\mu(c) = c$.
- $[\![P_1 \text{ UNION } P_2]\!]_G := [\![P_1]\!]_G \cup [\![P_2]\!]_G$.
- $[\![P_1 \text{ AND } P_2]\!]_G := [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$, where, $[\![P_1]\!]_G \bowtie [\![P_2]\!]_G = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]_G \text{ and } \mu_2 \in [\![P_2]\!]_G \text{ and } \mu_1 \sim \mu_2\}$. Two mappings μ_1 and μ_2 are *compatible*, denoted by $\mu_1 \sim \mu_2$, if they agree on the intersection of their domains.
- $[\![P_1 \text{ OPT } P_2]\!]_G := ([\![P_1]\!]_G \bowtie [\![P_2]\!]_G) \cup ([\![P_1]\!]_G \setminus [\![P_2]\!]_G)$, where $[\![P_1]\!]_G \setminus [\![P_2]\!]_G = \{\mu_1 \in [\![P_1]\!]_G \mid \neg\exists \mu_2 \in [\![P_2]\!]_G : \mu_1 \sim \mu_2\}$.
- $[\![P_1 \text{ MINUS } P_2]\!]_G := \{\mu_1 \in [\![P_1]\!]_G \mid \forall \mu_2 \in [\![P_2]\!]_G$, either $\mu_1 \not\sim \mu_2$ or $\mathrm{dom}\mu_1 \cap \mathrm{dom}\mu_2 = \emptyset\}$.
- $[\![P_1 \text{ BIND}_{?x}\,(c)]\!]_G := \{\mu \cup \{(?x \to c)\} \mid \mu \in [\![P_1]\!]_G\}$.
- $[\![(\mathrm{VALUES}\,\vec{W}\,D)]\!]_G := \{\mu \mid \mathrm{dom}(\mu) = \vec{W} \text{ and } \mu(\vec{W}) \in D\}$.
- $[\![P_1 \text{ FILTER } C]\!]_G := \{\mu \in [\![P_1]\!]_G \mid \mu \models C\}$, where the satisfaction of constraints EXISTS and NOT EXISTS are defined by a mapping μ:
  - $\mu \models \mathrm{EXISTS}(Q)$ if $[\![\mu(Q)]\!]_G \neq \emptyset$, where $\mu(Q)$ is a pattern obtained from Q by substituting c for $?x$ if $\mu(?x) = c$;
  - $\mu \models \mathrm{NOT\ EXISTS}(Q)$ if $\mu \not\models \mathrm{EXISTS}(Q)$.

Due to the limitation of space, the satisfaction of standard constraints in SPARQL 1.0 are omitted here but they can be defined in an intuitive way.

We say a pattern P is *satisfiable* if there exists a graph G such that $[\![P]\!]_G$ is nonempty. The (pattern) satisfiability problem of a fragment is to determine whether each pattern is satisfiable. The satisfiability problem for the full SPARQL 1.1 is undecidable (Pérez et al., 2009).

Undecidable Fragments of SPARQL 1.1

In this section, we present some undecidability results for the satisfiability problem of SPARQL 1.1 fragments. We first show that the satisfiability for $\mathcal{AM}$ patterns is undecidable by reducing it to the satisfiability problem for *Downward Algebra (DA)*, the algebra of finite binary relations with union, composition and difference, which is undecidable (Tan et al., 2014). Based on the undecidability of satisfiability for $\mathcal{AM}$ patterns, we identify some other fragments of SPARQL 1.1 that are undecidable.

Undecidability of $\mathcal{AM}$

In this section we show that the satisfiability problem of $\mathcal{AM}$ patterns is undecidable. However, it is not straightforward for reducing the satisfiability problem of DA to the pattern satisfiability of $\mathcal{AM}$ since DA is for describing binary

relations while RDF is about triples. Our approach is to employ a sub-fragment of $\mathcal{AM}$ containing so-called navigation patterns.

Theorem 1 *The problem of deciding whether a pattern is satisfiable in the SPARQL 1.1 fragment $\mathcal{AM}$ is undecidable.*

Note that (Zhang et al., 2016) is only able to prove that the pattern satisfiability of fragment $\mathcal{AMU}$ is undecidable. Their proof was done by expressing the complement of a pattern using UNION. However, it is unclear how this can be done without UNION. So, our theorem is a significant extension of the result for the undecidability of $\mathcal{AMU}$ in (Zhang et al., 2016).

In order to prove Theorem 1, we first introduce a fragment of $\mathcal{AM}$ called navigational $\mathcal{AM}$ patterns, denoted $\mathcal{AM}^{\mathrm{nav}}$, in which the complementation is closed. Then we prove the undecidability of $\mathcal{AM}^{\mathrm{nav}}$ by reducing satisfiability problem of the Downward Algebra (DA) to that of $\mathcal{AM}^{\mathrm{nav}}$ patterns (Tan et al., 2014). Then we conclude the undecidability of $\mathcal{AM}$ since $\mathcal{AM}^{\mathrm{nav}}$ is a fragment of $\mathcal{AM}$. By going to $\mathcal{AM}^{\mathrm{nav}}$, we are able to focus on binary relations for a pattern by projecting it on just two variables. In this way we can express the set of all possible pairs $(?x, ?y)$, where $?x$ occurs as the subject and $?y$ as the object for some triple.

The notion of *navigation patterns* is introduced to represent path queries via SPARQL in (Zhang and Van den Bussche, 2015). A navigation pattern is a triple $(P, ?x, ?y)$ where P is an $\mathcal{AM}$ pattern, $?x$ and $?y$ are variables occurring in P [1]. For instance, $((?x, p, ?y), ?x, ?y)$ is a navigation pattern while $((?x, p, ?z), ?x, ?y)$ is not a navigation pattern since $?y$ does not occur in the pattern $(?x, p, ?z)$.

Formally, given a navigation pattern P and an RDF graph G, they determine a binary relation $[\![(P, ?x, ?y)]\!]_G = \{(\mu(?x), \mu(?y)) \mid \mu \in [\![P]\!]_G\}$.

We now introduce the Downward Algebra (DA). DA-expressions are inductively defined as: $r \mid e \cup e \mid e - e \mid e \circ e$, where r is a binary relation symbol and e is a DA-expression.

Semantically, DA-expressions represent binary queries on binary relations, i.e., mappings from binary relations to binary relations. Let e be a DA-expression. For a binary relation J, the binary relation $e(J)$ is inductively defined as follows: (1) $r(J) = J$; (2) $(e_1 \cup e_2)(J) = e_1(J) \cup e_2(J)$; (3) $(e_1 - e_2)(J) = e_1(J) - e_2(J)$ (set difference); and (4) $(e_1 \circ e_2)(J) = \{(x, z) \mid \exists y : (x, y) \in e_1(J)$ and $(y, z) \in e_2(J)\}$. Here r is a relation symbol, e_1 and e_2 are DA-expressions.

A DA-expression e is *satisfiable* if there exists a finite binary relation J such that $e(J)$ is nonempty.

For instance, DA-expression $e = (r \circ r) - r$ is satisfiable since $e(J) = \{(a, c)\} (\neq \emptyset)$ for $J = \{(a, b), (b, c)\}$.

Two DA-expressions e and e' are *equivalent* if $e(J) = e'(J)$ for every relation J.

Given an RDF graph G, a binary graph $J(G)$ is defined by $(a, b) \in J(G)$ if and only if $(a, p, b) \in G$ for some $p \in I$.

[1] In this paper, navigation patterns are indeed safe patterns in the sense that $?x$ and $?y$ must occur in P (Zhang and Van den Bussche, 2015).

Lemma 2 *For each $\cup$-free DA-expression e, a navigation pattern $(P, ?x, ?y)$ in $\mathcal{AM}$ is constructed such that, for every RDF graph G, $e(J(G)) = [\![(P, ?x, ?y)]\!]_G$.*

Proof. We prove this lemma by induction on the structure of DA-expression e.

- If e is a relation name, we take $P = (?x, r, ?y)$. Then $(e)(J(G)) = [\![((?x, J, ?y), ?x, ?y)]\!]_G$ for each graph G.
- If e is of the form $e_1 - e_2$, then by induction, there exist two navigation patterns $(P_1, ?x_1, ?y_1)$ and $(P_2, ?x_2, ?y_2)$ for e_1 and e_2 such that $e_1(J(G)) = [\![(P_1, ?x_1, ?y_1)]\!]_G$ and $e_2(J(G)) = [\![(P_2, ?x_2, ?y_2)]\!]_G$, respectively.

  Let P_1' and P_2' be obtained from P_1 and P_2 by renaming the variables so that
  - $?x_1$ and $?x_2$ are renamed to $?x$;
  - $?y_1$ and $?y_2$ are renamed to $?y$; and
  - P_1' and P_2' have no common variables other than $?x, ?y$.

  Then $(P_1' \text{ MINUS } P_2', ?x, ?y)$ is a navigation pattern for e. Moreover, $e(J(G)) = (e_1 - e_2)(J(G)) = e_1(J(G)) - e_2(J(G)) = [\![(P_1, ?x_1, ?y_1)]\!]_G - [\![(P_2, ?x_2, ?y_2)]\!]_G = [\![(P_1' \text{ MINUS } P_2', ?x, ?y)]\!]_G$.
- If e is of the form $e_1 \circ e_2$, the proof is similar to the above case.

Lemma 2 can be extended to arbitrary DA-expressions.

Lemma 3 *For each DA-expression e, there exists a navigation pattern $(P, ?x, ?y)$ in $\mathcal{AM}$ such that, for every RDF graph G, $e(J(G)) = [\![(P, ?x, ?y)]\!]_G$.*

Proof. We observe the following rules for DA expressions:

- $(e_1 \cup e_2) \circ e_3 \to (e_1 \circ e_3) \cup (e_2 \circ e_3)$;
- $e_1 \circ (e_2 \cup e_3) \to (e_1 \circ e_2) \cup (e_1 \circ e_3)$;
- $(e_1 \cup e_2) - e_3 \to (e_1 - e_3) \cup (e_2 - e_3)$;
- $e_1 - (e_2 \cup e_3) \to (e_1 - e_2) - e_3$.

Thus, each DA-expresion e can be transformed into the union of some $\cup$-free DA-expressions $e_1 \cup \ldots \cup e_m$, where e_i is a $\cup$-free DA-expression for $i = 1, 2, \ldots, m$.

By Lemma 2, for each e_i $(i = 1, 2, \ldots, m)$, there exists a navigation pattern $(P_i, ?x, ?y)$ such that $e_i(J(G)) = [\![(P_i, ?x, ?y)]\!]_G$. Let $P = P_1 \text{ UNION} \ldots \text{UNION } P_m$. Then

$$\begin{aligned} e(J(G)) &= (e_1 \cup \ldots \cup e_m)(J(G)) \\ &= e_1(J(G)) \cup \ldots \cup e_m(J(G)) \\ &= [\![(P_1, ?x, ?y)]\!]_G \cup \cdots \cup [\![(P_m, ?x, ?y)]\!]_G \\ &= [\![(P_1 \text{ UNION} \ldots \text{UNION } P_m, ?x, ?y)]\!]_G \\ &= [\![(P, ?x, ?y)]\!]_G. \end{aligned}$$

The following lemma, which is a key for proving Theorem 1, shows that the union of some navigation patterns can be expressed by MINUS and AND. We recall that each pattern in $\mathcal{AMU}$ can be expressed in the UNION normal form $Q_1 \text{ UNION} \ldots \text{UNION } Q_m$, where each Q_i is UNION-free $(i = 1, \ldots, m)$ (Pérez et al., 2009), by these two equivalences: $P_1 \text{ MINUS } (P_2 \text{ UNION } P_3) \equiv (P_1 \text{ MINUS } P_2) \text{ MINUS } P_3$ and $(P_1 \text{ UNION } P_2) \text{ MINUS } P_3) \equiv (P_1 \text{ MINUS } P_3) \text{ UNION } (P_2 \text{ MINUS } P_3)$.

Lemma 4 *Let $(Q_i, ?x, ?y)$ be* UNION*-free navigation patterns in $\mathcal{AM}$ for $i = 1, \ldots, m$. Then, for any RDF graph G,*

$$[\![(Q_1 \text{ UNION} \ldots \text{UNION } Q_m, ?x, ?y)]\!]_G = [\![(\mathrm{Com}(Q_1, \ldots, Q_m), ?x, ?y)]\!]_G.$$

Here, $\mathrm{Com}(Q_1, \ldots, Q_m) = P_0$ MINUS $((P_1 \text{ MINUS } Q_1) \text{ AND} \ldots \text{AND } (P_m \text{ MINUS } Q_m))$,

and $P_i = (?x, ?t_i, ?u_i)$ AND $(?v_i, ?w_i, ?y)$ and $?t_i$, $?u_i$, $?v_i$, $?w_i$ are fresh variables ($i = 0, 1, 2, \ldots, m$).

Proof. Let P denote $(P_1 \text{ MINUS } Q_1) \text{ AND} \ldots \text{AND } (P_m \text{ MINUS } Q_m)$. Thus, $\mathrm{Com}(Q_1, \ldots, Q_m) = P_0$ MINUS P.

Let $(a, b) \in [\![(Q_1 \text{ UNION} \ldots \text{UNION } Q_m, ?x, ?y)]\!]_G$. Then there exists $i \in \{1, \ldots, m\}$ such that $(a, b) \in [\![(Q_i, ?x, ?y)]\!]_G$. We first note that $(a, b) \in [\![(P_0, ?x, ?y)]\!]_G$. By $(a, b) \in [\![(Q_i, ?x, ?y)]\!]_G$, we have $(a, b) \notin [\![(P_i \text{ MINUS } Q_i, ?x, ?y)]\!]_G$, which implies that $(a, b) \notin [\![(P, ?x, ?y)]\!]_G$. Then, $(a, b) \in [\![(\mathrm{Com}(Q_1, \ldots, Q_m), ?x, ?y)]\!]_G$

On the other hand, let $(a, b) \in [\![(\mathrm{Com}(Q_1, \ldots, Q_m), ?x, ?y)]\!]_G$. That is, $(a, b) \notin [\![(P_0 \text{ MINUS } P, ?x, ?y)]\!]_G$. We show that $(a, b) \in [\![(Q_i, ?x, ?y)]\!]_G$ for some $i \in \{1, \ldots, m\}$. By the assumption, $(a, b) \notin [\![(P, ?x, ?y)]\!]_G$. Then, $(a, b) \notin [\![(P_i \text{ MINUS } Q_i, ?x, ?y)]\!]_G$ for some $i \in \{1, \ldots, m\}$. Since $(a, b) \in [\![(P_0, ?x, ?y)]\!]_G$, we have that $(a, b) \in [\![(P_i, ?x, ?y)]\!]_G$. Thus, $(a, b) \in [\![(Q_i, ?x, ?y)]\!]_G$. Therefore, $(a, b) \in [\![(Q_1 \text{ UNION} \ldots \text{UNION } Q_m, ?x, ?y)]\!]_G$.

Proof of Theorem 1: For each DA-expression e, a navigation pattern $(P_e, ?x, ?y)$ can be constructed by

$$[\![(P_e, ?x, ?y)]\!]_G = e(J(G)).$$

This implies that the satisfiability problem of DA is reduced to that of navigation patterns in $\mathcal{AM}$. On the other hand, it is known that the satisfiability problem for DA-expressions is undecidable (Tan et al., 2014). Thus, the satisfiability problem of navigation patterns in $\mathcal{AM}$ is undecidable. This implies that the pattern satisfiability of $\mathcal{AM}$ is undecidable.

Other Undecidable Fragments

We have shown that the satisfiability problem of $\mathcal{AM}$ patterns is undecidable. Therefore, any fragments that are more expressive than $\mathcal{AM}$ will be undecidable too. In this subsection, we identify a few of such undecidable fragments of SPARQL 1.1.

The DIFF operator underlays the OPT operator in SPARQL although it is not really an operator of SPARQL. Semantically, let P_1, P_2 be two patterns and G be an RDF graph, $[\![P_1 \text{ DIFF } P_2]\!]_G = [\![P_1]\!]_G \setminus [\![P_2]\!]_G$.

Note that the DIFF operator is slightly different from the MINUS operator when domains of mappings of $[\![P_1]\!]_G$ and $[\![P_2]\!]_G$ are disjoint.

Indeed, we can rewrite MINUS to DIFF in a pattern where neither UNION nor OPT exists.

Let P be a pattern. We use $\mathrm{var}(P)$ to denote the collection of all variables occurring in P.

Let P be a pattern in $\mathcal{ABFMV}(\exists, \nexists)$. Let $\mathrm{MS}(P)$ be a set of variables defined as follows:

- $\mathrm{MS}(t) = \mathrm{var}(t)$ for any triple pattern t ;
- $\mathrm{MS}((\text{VALUES } \{?x_1, \ldots, ?x_m\}\ D)) = \{?x_1, \ldots, ?x_m\}$;
- $\mathrm{MS}(P_1 \text{ AND } P_2) = \mathrm{MS}(P_1) \cup \mathrm{MS}(P_2)$;
- $\mathrm{MS}(P_1 \text{ MINUS } P_2) = \mathrm{MS}(P_1)$;
- $\mathrm{MS}(P \text{ BIND}_{?x}\ (c)) = \mathrm{MS}(P) \cup \{?x\}$;
- $\mathrm{MS}(P \text{ FILTER } C) = \mathrm{MS}(P)$.

Lemma 5 *For any pattern P in $\mathcal{ABFMV}(\exists, \nexists)$, for any RDF graph G, for any $\mu \in [\![P]\!]_G$, $\mathrm{dom}(\mu) = \mathrm{MS}(P)$.*

Let P be a pattern in $\mathcal{ABFMV}(\exists, \nexists)$. We use $\delta(P)$ to denote a pattern obtained from P in a following way: for any subpattern Q of the form P_1 MINUS P_2 in P,

- rewrite P_1 MINUS P_2 to P_1 if $\mathrm{MS}(P_1) \cap \mathrm{MS}(P_2) = \emptyset$;
- rewrite P_1 MINUS P_2 to P_1 DIFF P_2 otherwise.

Note that $\delta(P)$ is in $\mathcal{ABDFV}(\exists, \nexists)$ (where $\mathcal{D}$ stands for DIFF).

By Lemma 5, we can conclude the following.

Proposition 6 *For any pattern P in $\mathcal{ABFMV}(\exists, \nexists)$, for any RDF graph G, $[\![P]\!]_G = [\![\delta(P)]\!]_G$.*

By Theorem 1 and Proposition 6, we can conclude that the satisfiability of fragment $\mathcal{AD}$ is undecidable since $\mathcal{AM}$ is already expressible in $\mathcal{AD}$. Since OPT envelops AND, the satisfiability of $\mathcal{MO}$ patterns is also undecidable.

Proposition 7 *The satisfiability problems of $\mathcal{AD}$ and $\mathcal{MO}$ patterns are undecidable.*

In order to see that the satisfiability for $\mathcal{AF}(\nexists)$ patterns is undecidable, we need the following lemma.

Lemma 8 DIFF *is expressible in $\mathcal{F}(\nexists)$.*

Clearly, we can conclude Lemma 8 by the following equivalence: P DIFF $Q \equiv P$ FILTER NOT EXISTS (Q) (Zhang et al., 2016; Kaminski et al., 2016).

By Lemma 8 and Theorem 1, we have the following.

Proposition 9 *The satisfiability problem for $\mathcal{AF}(\nexists)$ patterns is undecidable as well as $\mathcal{FO}$ patterns.*

Finally, both pattern satisfiability problems for two fragments $\mathcal{ABO}$ and $\mathcal{AOV}$ are undecidable. Indeed, it is easy to conclude that $\mathcal{AD}$ is expressible in both $\mathcal{ABOS}$ and $\mathcal{AOSV}$ by the equations: let $?x$ be a fresh variable and $a, b \in U$,

$$P \text{ DIFF } Q \equiv \mathrm{SELECT}_{\mathrm{var}(P)}((P \text{ OPT } (Q \text{ BIND}_{?x}\ (a))) \text{ AND } (P \text{ BIND}_{?x}\ (b))).$$

$$P \text{ DIFF } Q \equiv \mathrm{SELECT}_{\mathrm{var}(P)}((P \text{ OPT } (Q \text{ AND } (\text{VALUES } \{?x\}\ \{(a)\}))) \text{ AND } (\text{VALUES } \{?x\}\ \{(b)\})).$$

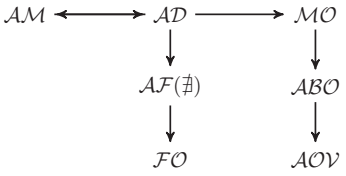
Thus their pattern satisfiability problems are undecidable. Since the SELECT operator does not affect the satisfiability, we conclude the following.

Corollary 10 *The satisfiability problem for $\mathcal{ABO}$ patterns is undecidable as well as $\mathcal{AOV}$ patterns.*

Note that OPT is necessary for the undecidability of $\mathcal{ABO}$ and $\mathcal{AOV}$. In other words, the satisfiability problem for $\mathcal{AB}$ patterns becomes decidable as well as $\mathcal{AV}$.

Thus, in a sense, we have identified all fragments of SPARQL 1.1 whose pattern satisfiability problems are undecidable.

The following diagram depicts all seven minimal fragments whose pattern satisfiability problem is undecidable. "$\mathcal{W}_1 \to \mathcal{W}_2$" means that $\mathcal{W}_1$ is expressible in $\mathcal{W}_2$.



Decidable Fragments

In this section, we show that the remaining fragments that are not discussed in the last two sections are decidable. These decidable fragments can be classified into the following three types:

- Fragments containing neither OPT nor AND.
- Fragments containing AND but no OPT.
- Fragments containing OPT and one of AND, BIND, UNION, and VALUES.

Decidability of the AND-OPT Free Fragment

The full AND-OPT free fragment is $\mathcal{BFMUV}(\exists, \nexists)$. We note that the two operators NOT EXISTS and UNION are redundant. Indeed, NOT EXISTS is expressible in $\mathcal{BMS}$ (Zhang et al., 2016): P FILTER NOT EXISTS$(Q) \equiv$ SELECT$_{\mathrm{var}(P)}((P \text{ BIND}_{?x}(c))\text{MINUS}(Q \text{ BIND}_{?x}(c)))$.

Also, in $\mathcal{BFMUV}(\exists, \nexists)$, each pattern is equivalent to a pattern in UNION normal form, which can be seen from these two equivalences:
(1) $(P_1 \text{UNION} P_2) \text{ BIND}_{?x}(c) \equiv (P_1 \text{BIND}_{?x}(c))\text{UNION}$ $(P_2 \text{ BIND}_{?x}(c))$ and
(2) P_1 FILTER EXISTS$(P_2$ UNION $P_3) \equiv (P_1$ FILTER EXISTS$(P_2))$ UNION $(P_1$ FILTER EXISTS$(P_3))$.

Thus, we need only to prove the decidability of $\mathcal{BFMV}(\exists)$. This can be done by reducing the satisfiability problem for $\mathcal{BFMV}(\exists)$ patterns to the satisfiability problem for guarded fragment of first-order logic, which is decidable (Grädel, 1999).

In this section, we work with first-order logic formulas over Σ with equality and the equality sign "$=$" is not an element of Σ.

For a first-order formula φ, $\varphi(?x_1, \ldots, ?x_k)$ indicates that the set of all free variables of φ is $\{?x_1, \ldots, ?x_k\}$.

Formally, the formulas of guarded fragment (GF) are generated by a recursive definition: (1) Atomic formulas of the form $?x = ?y$ and $?x = c$ are in GF for $c \in I \cup L$; (2) Relation atoms of the form $T(?x_1, ?x_2, ?x_3)$ are in GF; (3) If φ and ψ are formulas of GF, then so are $\neg\varphi$, $\varphi \vee \psi$, and $\varphi \wedge \psi$; and (4) If $\varphi(\bar{x}, \bar{y})$ is a formula of GF and $\alpha(\bar{x}, \bar{y})$ is a relation atom over Σ s.t. all free variables of φ occur in α, then $\exists\bar{y}(\alpha(\bar{x}, \bar{y}) \wedge \varphi(\bar{x}, \bar{y}))$ is a formula of GF.

Like in (Grädel, 1999), we allow equality and constants in GF.

As GF is a fragment of first-order logic, the semantics of GF is that of first-order logic. The semantics of GF is that of the relational calculus in database theory, interpreted over the active domain of the database (Abiteboul et al., 1995).

In the rest of this subsection we fix a relational vocabulary $\Sigma = \{T\}$ where T is a tertiary relation symbol. There is a strong correspondence between $\mathcal{BFMV}(\exists)$ and GF: one can be translated into the other.

Theorem 11 *For each $\mathcal{BFMV}(\exists)$ pattern P with $\mathrm{MS}(P) = \{?x_1, \ldots, ?x_k\}$, there exists a GF formula $\varphi_P(?x_1, \ldots, ?x_k)$ such that for every RDF graph G, for every $(d_1, \ldots, d_k) \in U^k$, $G \models \varphi_P(d_1, \ldots, d_k)$ iff $(?x_1 \to d_1, \ldots, ?x_k \to d_k) \in [\![P]\!]_G$.*

We note that the conclusion of the theorem for FILTER and DIFF cannot be directly obtained from definitions and thus the proof of this theorem is not straightforward. In our proof, we need a generalisation of (Leinders et al., 2005, Lemma 5) as the lemma does not assume the presence of equality and constants.

We first adapt some notions defined for relational algebra in (Leinders and Van den Buscche, 2007) to RDF.

Let G be an RDF graph. A set S is guarded in G if there exists some triple $(s, p, o) \in G$ such that $S \subseteq \{s, p, o\}$. For $C \subseteq I \cup L$, a set of elements X is C-stored in G if there exists a guarded set S in G such that $X \subseteq S \cup C$. A tuple $(d_1, \ldots, d_k)$ is C-stored in G if $\{d_1, \ldots, d_k\}$ is C-stored in G. Analogously, a mapping $(?x_1 \to d_1, \ldots, ?x_k \to d_k)$ is C-stored in G if $\{d_1, \ldots, d_k\}$ is C-stored in G.

By an easy induction on the structure of P, we can show the following lemma.

Lemma 12 *Let P be a pattern in $\mathcal{BFMV}(\exists)$ and G be an RDF graph. Then each mapping $\mu \in [\![P]\!]_G$ is C-stored in G where C is the set of all constants in P.*

Let $\mathrm{C} \subseteq I \cup L$. The set of C-stored k-tuples in structures of $\Sigma = \{T\}$ by the following formula (Grädel et al., 2002):

$$\mathbb{G}^{\mathrm{c}}(?x_1, \ldots, ?x_k) := \exists\bar{y}\,(T(\bar{y}) \wedge \bigwedge_{i=1}^{k}((\bigvee_{j} ?x_i = ?y_j) \vee (\bigvee_{c \in \mathrm{C}} ?x_i = c))).$$

Now we are ready to present the generalisation of (Leinders et al., 2005, Lemma 5) to GF with equality and constants.

Lemma 13 *If $\varphi(\bar{x}, \bar{y})$ is in GF with equality and constants, then $\exists\bar{y}(\mathbb{G}^{\mathrm{c}}(\bar{x}, \bar{y}) \wedge \varphi(\bar{x}, \bar{y}))$ is equivalent to a formula in GF.*

Proof of Theorem 11: We consider DIFF instead of MINUS in this proof since each pattern in $\mathcal{ABFMV}(\exists, \nexists)$ can be equivalently rewritten to a pattern in $\mathcal{ABDFV}(\exists, \nexists)$ by Proposition 6 and $\mathcal{BFMV}(\exists)$ is a fragment of $\mathcal{ABFMV}(\exists, \nexists)$. By induction on the structure of P.

- If P is a triple pattern of the form (u, v, w) then take $\varphi_P := T(u, v, w)$.
- If P is of the form $(\text{VALUES}\,\vec{W}\,D)$ where $\vec{W} = (?x_1, \ldots, ?x_k)$ and $D = \{(c_1^1, \ldots, c_k^1), \ldots, (c_1^m, \ldots, c_k^m)\}$ then take $\varphi_P(x_1, \ldots, x_k) := (\bigvee_{i=1}^{m} \bigwedge_{j=1}^{k} x_j = c_j^i)$.
- If P is of the form P_1 $\text{BIND}_{?x}$ (c) then, by induction, we have a formula $\varphi_{P_1}(?x_1, \ldots, ?x_k)$. Now, $\varphi_P := \varphi_{P_1}(?x_1, \ldots, ?x_k) \wedge ?x = c$.
- If P is of the form P_1 FILTER C then, by induction, we have a formula $\varphi_{P_1}(?x_1, \ldots, ?x_k)$. Take $\varphi_P := \varphi_{P_1}(?x_1, \ldots, ?x_k) \wedge C$.
- If P is of the form P_1 FILTER EXISTS(P_2) then, by induction, we have formulas $\varphi_{P_1}(\bar{x}, \bar{y})$ and $\varphi_{P_2}(\bar{y}, \bar{z})$ where $\bar{y} = \text{var}(P_1) \cap \text{var}(P_2)$. We show how to obtain a GF formula equivalent to the formula $\varphi_{P_1}(\bar{x}, \bar{y}) \wedge \exists \bar{z}\, \varphi_{P_2}(\bar{y}, \bar{z})$. By Lemma 12, $\varphi_{P_2}(\bar{y}, \bar{z})$ is equivalent to the formula $\mathbb{G}^c(\bar{y}, \bar{z}) \wedge \varphi_{P_2}(\bar{y}, \bar{z})$. Then ϕ is equivalent to the formula $\varphi_{P_1}(\bar{x}, \bar{y}) \wedge \exists \bar{z}(\mathbb{G}_n(\bar{x}, \bar{z}) \wedge \varphi_{P_2}(\bar{x}, \bar{z}))$. By Lemma 13, there exists some formula $\varphi'_{P_2}(\bar{y})$ in GF such that $\varphi'_{P_2}(\bar{y})$ is equivalent to $\exists \bar{z}(\mathbb{G}_n(\bar{x}, \bar{z}) \wedge \varphi_{P_2}(\bar{x}, \bar{z}))$ since $\varphi_{P_2}(\bar{y}, \bar{z})$ is in GF. Now, $\varphi_P(\bar{x}, \bar{y})$ is the formula $\varphi_{P_1}(\bar{x}, \bar{y}) \wedge \varphi'_{P_2}(\bar{y})$.
- Finally, if P is of the form P_1 DIFF P_2 then, by induction, we have formulas $\varphi_{P_1}(\bar{x}, \bar{y})$ and $\varphi_{P_2}(\bar{y}, \bar{z})$ where $\bar{y} = \text{var}(P_1) \cap \text{var}(P_2)$. We show how to obtain a GF formula equivalent to the formula $\varphi_{P_1}(\bar{x}, \bar{y}) \wedge \neg\exists \bar{z}\, \varphi_{P_2}(\bar{y}, \bar{z})$. By Lemma 12, $\varphi_{P_2}(\bar{y}, \bar{z})$ is equivalent to the formula $\mathbb{G}^c(\bar{y}, \bar{z}) \wedge \varphi_{P_2}(\bar{y}, \bar{z})$. Then ϕ is equivalent to the formula $\varphi_{P_1}(\bar{x}, \bar{y}) \wedge \exists \bar{z}(\mathbb{G}_n(\bar{x}, \bar{z}) \wedge \varphi_{P_2}(\bar{x}, \bar{z}))$. By Lemma 13, there exists some formula $\varphi'_{P_2}(\bar{y})$ in GF such that $\varphi'_{P_2}(\bar{y})$ is equivalent to $\exists \bar{z}(\mathbb{G}_n(\bar{x}, \bar{z}) \wedge \varphi_{P_2}(\bar{x}, \bar{z}))$ since $\varphi_{P_2}(\bar{y}, \bar{z})$ is in GF. Now, $\varphi_P(\bar{x}, \bar{y})$ is the formula $\varphi_{P_1}(\bar{x}, \bar{y}) \wedge \neg\varphi'_{P_2}(\bar{y})$.

By Theorem 11, it follows the decidability of $\mathcal{BFMV}(\exists)$.

Theorem 14 *The satisfiability problems for $\mathcal{BFMV}(\exists)$ and $\mathcal{BFMUV}(\exists, \sharp)$ patterns are decidable.*

Since $\mathcal{BFMUV}(\exists, \sharp)$ is also expressible in GF where UNION can be expressed by the disjunction connective $\vee$ of GF and the satisfiability problem for GF is in EXPTIME (Grädel, 1999), the satisfiability problem for $\mathcal{BFMUV}(\exists, \sharp)$ is EXPTIME.

Moreover, we can show that the satisfiability problem for $\mathcal{BFMUV}(\exists, \sharp)$ patterns is EXPTIME-hard by reducing the satisfiability problem for *semijoin algebra*, denoted by SA^2, with a single binary relation symbol (Leinders et al., 2005) to the satisfiability problem of $\mathcal{BFMUV}(\exists, \sharp)$ patterns.

Proposition 15 *The satisfiability problem for $\mathcal{BFMUV}(\exists, \sharp)$ patterns is EXPTIME-Complete.*

Proof. (Sketch) For each SA^2-expression e, we can construct a pattern P_e in $\mathcal{FMU}(\exists)$ in polynomial time s.t. e is satisfaible iff P_e is satisfiable. Since the satisfiability problem for SA^2-expressions is EXPTIME-hard (Leinders et al., 2005, Theorem 11), the satisfiability problem for $\mathcal{FMU}(\exists)$ patterns is EXPTIME-hard. That is, the satisfiability problem for $\mathcal{BFMUV}(\exists, \sharp)$ patterns is EXPTIME-Complete.

Decidable OPT-Free Fragments

We show that the satisfiability for patterns in the largest OPT-free fragment $\mathcal{ABFUV}(\exists)$ is decidable.

Note that EXISTS can be expressed by AND and SELECT (Zhang et al., 2016) : P FILTER EXISTS$(Q) \equiv$ $\text{SELECT}_{\text{var}(P)}(P$ AND $Q)$.

Moreover, each pattern in $\mathcal{ABFUV}(\exists)$ is equivalent to a pattern in UNION normal form.

Thus, it suffices to show the decidability of satisfiability for $\mathcal{ABFV}$ by checking the satisfiability of bound constraints. We consider only constraints of the form $?x = c$, $?x \neq c$, $?x = ?y$, and $?x \neq ?y$ since $\text{bound}(?x)$ and $\neg\text{bound}(?x)$ can be easily removed by syntactical rewriting. In addition, we consider only safe patterns since unsafe patterns are always unsatisfiable (Pérez et al., 2009). A pattern is *safe* if for every subpattern of the form P FILTER C, variables of C must occur in P.

To check the satisfiability of bound constraints, we associate each pattern P with a pair $\Gamma(P) = (S, \mathcal{C})$ where S is a set of variables and $\mathcal{C}$ is a set of atomic constraints defined as follows.

- $\Gamma((u, v, w)) = (\{u, v, w\} \cap V, \emptyset)$;
- $\Gamma((\text{VALUES}\{(?x_1, \ldots, ?x_k)\}, \{(c_1, \ldots, c_k)\})) := (\{?x_1, \ldots, ?x_n\}, \{?x_i = c_i \mid i = 1, 2, \ldots, k\})$;
- $\Gamma(P_1 \text{ AND } P_2) := (S_1 \cup S_2, \mathcal{C}_1 \cup \mathcal{C}_2)$;
- $\Gamma(P_1 \text{ BIND}_{?x}\ (c)) := (S_1 \cup \{?x\}, \mathcal{C}_1 \cup \{?x = c\})$;
- $\Gamma(P_1 \text{ FILTER } C) := (S_1, \mathcal{C}_1 \cup \{C\})$.

Here $\Gamma(P_i) = (S_i, \mathcal{C}_i)$ $(i = 1, 2)$.

We now establish the main result of this subsection.

Theorem 16 *Let P be an $\mathcal{ABFV}$ pattern and $\Gamma(P) = (S, \mathcal{C})$. Then P is satisfiable if and only if $\mathcal{C}$ is satisfiable.*

The "only-if" direction of Theorem 16 can be shown by Lemma 17.

Lemma 17 *Let P be an $\mathcal{ABFV}$ pattern and G a graph. If $\mu \in [\![P]\!]_G$, then $\mu \models C$ for any constraint $C \in \mathcal{C}$.*

The "if" direction of Theorem 16 for $\mathcal{ABFV}$ follows by Lemma 18.

Lemma 18 *Let P be an $\mathcal{ABFV}$ pattern, $\Gamma(P) = (S, \mathcal{C})$, μ a mapping from $\text{var}(\mathcal{C})$ to U with $\mu \models C$ for every $C \in \mathcal{C}$, $c \in I$ a constant that does not appear in any filter condition in P, and G the RDF graph $\{(\bar{\mu}(u), \bar{\mu}(v), \bar{\mu}(w)) \mid (u, v, w) \in P\}$. Then $\bar{\mu} \in [\![P]\!]_G$, where*

$$\bar{\mu}(?x) = \begin{cases} \mu(?x), & ?x \in \text{var}(\mathcal{C}); \\ c, & ?x \in \text{var}(P) \setminus \text{var}(\mathcal{C}). \end{cases}$$

Proof. By induction on the structure of P.

- If P is a triple pattern (u, v, w) then $S = \{u, v, w\} \cap V$. Since $\mathcal{C} = \emptyset$, μ is empty. Thus $(\bar{\mu}(u), \bar{\mu}(v), \bar{\mu}(w)) \in G$, we have $\bar{\mu} \in [\![P]\!]_G$.

- If P is a VALUES pattern of the form $(\text{VALUES}\,\vec{W}\,D)$ where $\vec{W} = (?x_1, \ldots, ?x_k)$ and $D = \{(c_1, \ldots, c_k)\}$ then $S = \{?x_1, \ldots, ?x_k\}$ and $\mathcal{C} = \{?x_i = c_i \mid i = 1, 2, \ldots, k\}$. Let $\mu = (?x_1 \to c_i, \ldots, ?x_k \to c_k)$. For any graph G containing all possible triples $(\mu(u), \mu(v), \mu(w))$, we have $\mu \in [\![P]\!]_G$. Then μ is desired.
- If P is of the form $P_1 \,\text{AND}\, P_2$, then we have $S = S_1 \cup S_2$ and $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ with $\Gamma(P_i) = (S_i, \mathcal{C}_i)$ for $i = 1, 2$. Let μ be a mapping from $\text{var}(\mathcal{C})$ to U such that $\mu \models C$ for any constraint $C \in \mathcal{C}$. Then let $\bar{\mu}$ be a mapping from $\text{var}(P)$ to U constructed as above and G be the RDF graph containing all possible triples $(\bar{\mu}(u), \bar{\mu}(v), \bar{\mu}(w))$ where (u, v, w) is a triple pattern in P. By induction, $\bar{\mu}|_{S_1} \in [\![P_1]\!]_G$ and $\bar{\mu}|_{S_1} \in [\![P_1]\!]_G$ (Note that $\mu|_S$ is the restriction of μ under S.) Clearly $\bar{\mu}|_{S_1} \sim \bar{\mu}|_{S_2}$ since they are restrictions of the same mapping. Hence $\bar{\mu}|_{S_1} \cup \bar{\mu}|_{S_2} = \mu_{S_1 \cup S_2} \in [\![P]\!]_G$.
- If P is of the form $P_1 \,\text{BIND}_{?x}\,(a)$, then we have $S = S_1 \cup \{?x\}$ and $\mathcal{C} = \mathcal{C}_1 \cup \{?x = a\}$ with $\Gamma(P_1) = (S_1, \mathcal{C}_1)$. Let μ be a mapping from $\text{var}(\mathcal{C})$ to U such that $\mu \models C$ for any constraint $C \in \mathcal{C}$. Clearly, $\mu(?x) = a$. Then let $\bar{\mu}$ be a mapping from $\text{var}(P)$ to U constructed as above and G be the RDF graph containing all possible triples $(\bar{\mu}(u), \bar{\mu}(v), \bar{\mu}(w))$ where (u, v, w) is a triple pattern in P. By induction, $\bar{\mu}|_{S_1} \in [\![P_1]\!]_G$. Clearly $\bar{\mu}|_{S_1} \sim \{?x \to a\}$ since $?x \notin S_1$ by our restriction (Otherwise, P is unsatisfiable). Hence $\bar{\mu}|_{S_1} \cup \{?x \to a\} = \mu_{S_1 \cup \{?x\}} \in [\![P]\!]_G$.
- Finally, if P is of the form $P_1 \,\text{FILTER}\, C$, then we know that $S = S_1$ and $\mathcal{C} = \mathcal{C}_1 \cup \{C\}$ with $\Gamma(P_1) = (S_1, \mathcal{C}_1)$. Let μ be a mapping from $\text{var}(\mathcal{C})$ to U such that $\mu \models C$ for any constraint $C \in \mathcal{C}$. Clearly, $\mu \models C$. Then let $\bar{\mu}$ be a mapping from $\text{var}(P)$ to U constructed as above and G be the RDF graph containing all possible triples $(\bar{\mu}(u), \bar{\mu}(v), \bar{\mu}(w))$ where (u, v, w) is a triple pattern in P. By induction, $\bar{\mu}|_{S_1} \in [\![P_1$. Since $\bar{\mu}(?x) = \mu(?x)$ for any $?x \in \text{var}(\mathcal{C}) \subseteq S_1$, $\mu \models C$ implies $\bar{\mu} \models C$. Then $\bar{\mu}|_{S_1} \in [\![P]\!]_G$.

By Theorem 16, we have the following result.

Corollary 19 *The satisfiability problem for $\mathcal{ABFUV}(\exists)$ patterns is decidable.*

Since the satisfiability problem for $\mathcal{ABFU}$ (a subfragment of $\mathcal{ABFUV}(\exists)$) is actually NP-hard (Zhang et al., 2016, Proposition 10), we have the following result.

Corollary 20 *The satisfiability problem for $\mathcal{ABFUV}(\exists)$ patterns is NP-hard.*

Decidable Fragments Containing OPT

We have shown that a fragment containing OPT is undecidable if it also contains one of the following sets of operators: $\{\text{MINUS}\}$, $\{\text{AND}, \text{BIND}\}$, $\{\text{AND}, \text{VALUES}\}$, and $\{\text{FILTER}, \text{EXISTS}, \text{NOT EXISTS}\}$. In this subsection, we show that the remaining fragments containing OPT are decidable. Specifically, a fragment containing OPT is decidable if it contains one of AND, UNION, BIND, and VALUES. In fact, we come up with a stronger result than this by showing that the pattern satisfiability problems for both $\mathcal{AOU}$ and $\mathcal{BOUV}$ are decidable.

We note that $\mathcal{AOU}$ is a fragment of SPARQL$(\text{bound}, =, \neq_c)$ (Zhang et al., 2016) without FILTER operator. Indeed, if no triple pattern contains any wrong literals, then every $\mathcal{AOU}$ pattern is satisfiable since the mapping from the set of variables to a single constant is a solution. On the other hand, it is proven that the pattern satisfiability for SPARQL$(\text{bound}, =, \neq_c)$ without FILTER is decidable (Zhang et al., 2016). Thus, the pattern satisfiability for $\mathcal{AOU}$ is also decidable.

However, the decidability of $\mathcal{BOUV}$ patterns is not straightforward. In order to prove this decidability result, we reduce the satisfiability problem for $\mathcal{BOUV}$ patterns to the problem of deciding whether a triple pattern contains a wrong literal.

To do so, we introduce the notion of *principal subpattern*. The *principal subpattern* of a pattern P is an OPT-free subpattern of P, written by $\text{ps}(P)$, defined as follows: (1) $\text{ps}((u, v, w)) := (u, v, w)$; (2) $\text{ps}(P_1 \,\text{BIND}_{?x}\,(c)) := \text{ps}(P_1) \,\text{BIND}_{?x}\,(c)$; (3) $\text{ps}(\text{VALUES}\,\vec{W}\,D) = (\text{VALUES}\,\vec{W}\,D)$; (4) $\text{ps}(P_1 \,\text{UNION}\, P_2) := \text{ps}(P_1) \,\text{UNION}\, \text{ps}(P_2)$; and (5) $\text{ps}(P_1 \,\text{OPT}\, P_2) := \text{ps}(P_1)$.

Now, we will show that the satisfiability problem for $\mathcal{BOUV}$ can be reduced to the satisfiability problem for $\mathcal{BUV}$ patterns.

Proposition 21 *A $\mathcal{BOUV}$ pattern P is satisfiable if and only if $\text{ps}(P)$ is satisfiable.*

By Proposition 21, we can determine whether a $\mathcal{BOUV}$ pattern is satisfiable by checking the satisfiability of its principal subpattern which is in $\mathcal{BO}$. Note that principal subpatterns are in $\mathcal{BUV}$, that is, they are built on either triple patterns or VALUES patterns with the BIND operator. Since each variable to be binded is fresh, the satisfiability problem for $\mathcal{BV}$ patterns can be reduced to the satisfiability problem for $\mathcal{V}$ patterns. Normally, besides VALUES patterns, all triple patterns are satisfiable except for triple patterns with wrong literals (Zhang et al., 2016). Thus, the satisfiability problem for $\mathcal{BOUV}$ patterns can be reduced to the problem of checking triple patterns with wrong literals. This is similar to the case of $\mathcal{AOU}$ where patterns in $\mathcal{AU}$ are always satisfiable except for triple patterns with wrong literals (Zhang et al., 2016).

In short, the satisfiability problem of $\mathcal{AOU}$ patterns and $\mathcal{BOUV}$ patterns can be reduced to the problem whether a triple pattern contains a wrong literal, which can be checked in linear time. Therefore, we have the following result.

Corollary 22 *The satisfiability problems for both $\mathcal{AOU}$ and $\mathcal{BOUV}$ patterns can be decided in linear time.*

Conclusion

In this paper we have shown that the satisfiability problem of the fragment $\mathcal{AM}$ patterns is undecidable. This is interesting as it is unclear how to express the complement of a pattern by AND and MINUS without UNION. We have also shown the pattern decidability for several fragments including $\mathcal{BFMUV}(\exists, \nexists)$, $\mathcal{ABFUV}(\exists)$, $\mathcal{AOU}$, and

$\mathcal{BOUV}$. Based on these results, we are able to paint a complete picture for decidability/undecidability of all 640 fragments of SPARQL 1.1 patterns and three classes of decidable fragments with different computational complexity. As well as their theoretical interest, our results provide a guideline for designing SPARQL query languages and implementing them.

Note that OPT_F is proposed in (Kontchakov and V. Kostylev, 2016; Kaminski et al., 2017) as LeftJoin (a more expressive operator) to replace OPT in the general SPARQL. Most of our results can be generalised to SPARQL fragments containing OPT_F if F belongs to one of the classes of filters for which the satisfiability is decidable (Zhang et al., 2016). However, if general filter expressions are allowed, further investigation would be needed. In addition, we plan to conduct a complexity analysis of SPARQL 1.1 language with some other important features such as aggregation functions which possibly cause unsatisfiability (Han et al., 2016).

Acknowledgments

This work is supported by the National Key Research and Development Program of China (2016YFB1000603), the National Natural Science Foundation of China (61502336), and the Key Technology Research and Development Program of Tianjin (16YFZCGX00210).

References

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Andréka, H., Givant, S., and Németi, I. 1997. *Decision Problems for Equational Theories of Relational Algebras*, *Memoirs* 126. American Mathematical Society.

Andréka, H., Hodkinson, I., and Németi, I. 1999. Finite Algebras of Relations are Representable on Finite Sets. *J. Symb. Log.* 64(1): 243–267.

Andréka, H., Németi, I., and van Benthem, J. 1998. Modal Languages and Bounded Fragments of Predicate Logic. *J. Philosophical Logic* 27(3): 217–274.

Angles, R. and Gutierrez, C. 2008. The Expressive Power of SPARQL. In *Proceedings of ISWC'08*, 114–129. Springer.

Arenas, M., Conca, S., and Pérez, J. 2012. Counting Beyond a Yottabyte, or How SPARQL 1.1 Property Paths will Prevent Adoption of the Standard. In *Proceedings of WWW'12*, 629–638. ACM.

Arenas, M. and Pérez, J. 2011. Querying Semantic Web Data with SPARQL. In *Proceedings of PODS'11*, 305–316. ACM.

Barany, V., ten Cate, B., and Otto, M. 2012. Queries with Guarded Negation. *PVLDB* 5(11): 1328–1339.

Grädel, E. 1999. On the Restraining Power of Guards. *J. Symb. Log.* 64(4): 1719–1742.

Grädel, E., Hirsch, C., and Otto, M. 2002. Back and Forth between Guarded and Modal Logics. *ACM Trans. Comput. Log.* 3(3): 418–463.

Han, X., Feng, Z., Zhang, X., Wang, X., Rao, G., and Jiang, S. 2016. On the Statistical Analysis of Practical SPARQL Queries. In *Proceedings of WebDB'16*, Article 2. ACM.

Harris, S. and Seaborne, A. 2013. SPARQL 1.1 Query Language. *W3C Recommendation*.

Kaminski, M., Kostylev, E., and Cuenca Grau, B. 2016. Semantics and Expressive Power of Subqueries and Aggregates in SPARQL 1.1. In *Proceedings of WWW'16*, 227–238. ACM.

Kaminski, M., Kostylev, E., and Cuenca Grau, B. 2017. Query Nesting, Assignment, and Aggregation in SPARQL 1.1. *ACM Trans. Database Syst.* 42(3)(17): 1–46.

Kontchakov, R. and V. Kostylev, E. 2016. On Expressibility of Non-monotone Operators in SPARQL. In *Proceedings of KR'16*, 369–379. AAAI Press.

Kostylev, E., Reutter, J., Romero, M., and Vrgoč, D. 2015. SPARQL with Property Paths. In *Proceedings of ISWC'15*, 3–18. Springer.

Kroening, D. and Strichman, O. 2008. *Decision procedures*. Springer.

Leinders, D. and Van den Bussche, J. 2007. On the Complexity of Division and Set Joins in the Relational Algebra. *J. Comput. Syst. Sci.* 73(4): 538–549.

Leinders, D., Marx, M., Tyszkiewicz, J., and Van den Bussche, J. 2005. The Semijoin Algebra and the Guarded Fragment. *J. Log. Lang. Inf.* 14(3): 331–343.

Letelier, A., Pérez, J., Pichler, R., and Skritek, S. 2013. Static Analysis and Optimization of Semantic Web Queries. *ACM Trans. Database Syst. 38*(4), Article 25.

Pérez, J., Arenas, M., and Gutierrez, C. 2009. Semantics and Complexity of SPARQL. *ACM Trans. Database Syst.*, *34*(3), article 16.

Polleres, A.(2007). From SPARQL to Rules (and Back). In *Proceedings of WWW'07*, 787–796. ACM.

Prud'hommeaux, E. and Seaborne, A. 2008. SPARQL Query Language for RDF. *W3C Recommendation*.

Schmidt, M., Meier, M., and Lausen, G. 2010. Foundations of SPARQL Query Optimization. In *Proceedings of ICDT'10*, 4–33.

Tan, T., Van den Bussche, J., and Zhang, X. 2014. Undecidability of Satisfiability in the Algebra of Finite Binary Relations with Union, Composition, and Difference. *arXiv*:1406.0349.

Wudage, M., Euzenat, J., Genevès, P., and Layaïda, N. 2012. SPARQL Query Containment under SHI Axioms. In *Proceedings of AAAI'12*, 10–16. AAAI Press.

Zhang, X., Van den Bussche, J., and Picalausa, F. 2016. On the Satisfiability Problem for SPARQL Patterns. *J. Artif. Intell. Res.* 56: 403-428.

Zhang, X. and Van den Bussche, J. 2015. On the Power of SPARQL in Expressing Navigational Queries. *Computer J.* 58 (11): 2841-2851.

Zhang, X. and Van den Bussche, J. 2014. On the Primitivity of Operators in SPARQL. *Inf. Process. Lett.*, 114(9): 480–485.