On the Power of SPARQL in Expressing Navigational Queries

XIAOWANG ZHANG AND JAN VAN DEN BUSSCHE

Hasselt University and transnational University of Limburg, 3500 Hasselt, Belgium
Email: xiaowang.zhang@uhasselt.be, jan.vandenbussche@uhasselt.be

Navigational queries on graph databases return binary relations over the nodes of the graph. The calculus of relations, popularized by Tarski, serves as a natural benchmark for first-order navigational querying. Recently, nested regular expressions (nre's) have been proposed to extend navigational querying to RDF graphs, i.e., ternary relations. This paper investigates the expressiveness of nre's and their relationship to basic SPARQL queries. An elegant proof is given to the effect that nre queries are already expressible as basic SPARQL queries. This result takes exception of nre's involving Kleene star (transitive closure), but on the other hand it holds even when extending nre's with negation (complementation). The resulting language of "star-free nre's with negation (sfnre$^\neg$)" can in fact be captured by a precisely delineated fragment of SPARQL, called Tarski-SPARQL. The resulting language is also compared with an alternative extension that adds negation in the form of the difference operator. While sfnre$^\neg$ queries are subsumed by first-order logic with 3 variables (FO^3), it is shown that some natural FO^3 queries are not expressible in nre$^\neg$, even when allowing transitive closure.

Keywords: RDF databases; nested regular expression; star-free nested regular expression; complement; difference; Tarski-SPARQL

1. INTRODUCTION

Graph databases, as well as the design and analysis of query languages appropriate for graph data, have a rich history in database systems and theory research [1]. Originally investigated from the perspective of object-oriented databases, interest in graph databases research has been continually renewed, motivated by data on the Web [2, 3], new applications such as dataspaces [4], and new infrastructures such as Linked Data [5].

Information on the Semantic Web is popularly represented in RDF [6], which is again a graph-oriented data model. The standard language for querying RDF data is SPARQL [7]. The current version 1.1 of SPARQL extends the previous version 1.0 with many features such as negation, property path expressions, assignment, aggregation, subqueries, federation, and updates. When extending an important language, it is equally important to understand the need for this extension, as well as the circumstances where the extension is actually unnecessary. Such is the theme of the present paper, where we focus on proposed extensions to extend the *navigational* capabilities of SPARQL.

The navigational nature of accessing data is indeed one of the main characteristics of graph query languages. Navigation over binary relational structures, which encompasses both trees and graphs, can be naturally formalized in terms of expressions built up from relation names using a set of essential operators on binary relations [8, 9, 10, 11, 12]. These operators are union; difference; composition; inversion; projection (also called *node test*); the identity relation; and transitive closure. Interestingly, with the exception of transitive closure, these operators can be traced way back to the *calculus of relations* created by Peirce and Schröder, and popularized and greatly developed by Tarski and his collaborators [13, 14, 15]. For this reason, navigational query expressions on graph databases have also been referred to as "Tarski expressions" [16, 17].

There is a hurdle, however, to applying the calculus of relations straight away to RDF graphs: the latter are ternary, rather than binary, relations. This hurdle was overcome by Pérez et al. [18] who defined the formalism of *nested regular expressions (nre's)* as an elegant adaptation of the calculus of relations to RDF graphs. Apart from node tests, nre's are comprised of the operators union, composition, inversion, transitive closure, and the identity relation. These operators are applied starting from the three possible binary relations (each called an *axis*) that are projections of the ternary relation. Node tests, which provide explicit projection on each of the three axes, then provide the nested aspect of nre's.

Pérez et al. actually proposed to extend SPARQL 1.0 to nSPARQL by allowing nre's in the place of triple patterns. nSPARQL influenced the SPARQL 1.1 standard [19] in that nre's have been partly incorporated into SPARQL 1.1, in the form of *property path expressions*. Node tests are not supported, however, and property path expressions have not been given the clean compositional semantics of nre's. For these reasons, in this paper, we continue to investigate the original nre's and not the SPARQL property paths. We also cite the work by Alkhateeb et al. [20] on adding path querying capabilities to SPARQL, noting at the same time

that the focus of the present paper is mostly not on recursive path querying.

In this paper we offer the following contributions.

Two forms of negation. The classical calculus of relations includes negation in the form of the complementation operator. Correspondingly we add complementation to nre's, obtaining *nre's with negation (nre$^\neg$)*. An alternative way to extend nre's with negation is to add set difference instead of complementation, resulting in the language nre$^-$. We point out that nre$^-$ is, in a sense, more desirable than nre$^\neg$, since nre$^-$ preserves a natural connectivity property of nre's, which is lost in nre$^\neg$.

From star-free nre$^\neg$'s to basic SPARQL. Considering *star-free* nre$^\neg$'s, we show how these can be translated into basic SPARQL queries. Star-freeness means that the Kleene star operator (transive closure) is not used in the expression. We thus show that nre's, even with negation, only add something to basic SPARQL because of the transitive closure operator. This observation is in a sense obvious, because star-free nre$^\neg$'s are clearly expressible in first-order logic, and it has been known for some time [21, 22, 23] that basic SPARQL can express first-order queries. (A similar remark was made by Alkhateeb [24].) Nevertheless we believe our contribution is interesting for the following reasons.

1. We give an elegant and direct translation, without the detour via first-order logic, based on a new auxiliary notion of *navigation pattern*.
2. A translation of property paths to basic SPARQL expressions is also described in SPARQL 1.1 [7], but that translation leaves much to be desired. It is awkwardly formalized, in a non-compositional manner, is cumbersome to read, and its formal correctness is unclear. Our translation is compositional, encompasses a broader range of expressions (SPARQL 1.1 property paths do not have node tests or negation, for instance), and the formal correctness is evident. In this way our contribution improves the current state of the art.
3. The transitive closure operator, while fundamental for graph querying, is not always indispensable. It is not expressible in the relational algebra, which is the baseline level of expressiveness efficiently supported by database query processors [25]. In natural networks, the diameter is small [26, 27] so transitive closure could be replaced by a fixed number of compositions. In the same vein, the Facebook Query Language (FQL) [28] does not include transitive closure. For all these reasons it is interesting to understand the expressiveness of star-free navigational querying in as much detail as possible.
4. Although star-free nre$^\neg$'s are expressible in basic SPARQL, it is for many people much more attractive and natural to express queries in navigational style [11, 12, 29]. Also XPath queries belong to this style of querying, and it is noteworthy that Versa, one of the earliest RDF query language proposals, was explicitly XPath-based. By our direct translation, nre-style queries can then be immediately supported by any basic SPARQL query processor.
5. Finally, our translation is so transparant that it entails a syntactically delineated fragment of SPARQL, which we baptize "Tarski-SPARQL", exactly equivalent to star-free nre$^\neg$.

We note that our terminology of star-free nre is inspired by the known notion of star-free regular expression, similarly obtained from standard regular expressions by disallowing Kleene star but adding in complementation [30, 31].

Connection with FO^3. In view of the classical equivalence (for binary relation queries) between the calculus of relations and FO^3, the three-variable fragment of first-order logic [14, 32], we look at the relationship between nre$^\neg$ and FO^3 in the RDF context. While nre$^\neg$ is still subsumed by FO^3, it turns out to be not longer equivalent, and we give a very natural counterexample with a simple proof of inexpressibility. In this connection we also cite the work by Vrgoč et al. [29] who defined an RDF query language, called TriAL$^=$, sitting in expressiveness between FO^3 and FO^4.

We believe our work is interesting also because the calculus of relations is part not only of graph query languages, but also of a wide variety of logics in computer science, such as description logics (in the form of *role expressions*), dynamic logics (in the form of *programs*), arrow logics, and relation algebras [33, 34, 32, 35, 36]. Hence our work explicitly connects SPARQL to these other fields.

The further contents of this paper may be summarized as follows. In Section 2 we recall the basic notions concerning SPARQL patterns and queries. In Section 3 we define nre's and their extension with negation. Section 4 we give the translation from star-free nre$^\neg$'s into basic SPARQL queries. Further observations on the expressive power of nre$^\neg$'s are offered in section 5. In particular we can express the *residual* operators [15]. These operators provide the counterpart of universal quantification [37] for graph queries, thus allowing the expression of interesting decision support queries over RDF graphs. We conclude in Section 6.

2. SPARQL

In this section, we recall the syntax and semantics of SPARQL 1.0, largely following the excellent expositions [38, 39].

RDF graphs Let I, B, and L be infinite countable sets of *IRIs*, *blank nodes* and *literals*, respectively. These three sets are pairwise disjoint. We denote the union $I \cup B \cup L$ by U, and elements of $I \cup L$ will be referred to as *constants*. For the purposes of this paper, the distinction between IRIs and literals will not be important.

A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*. An *RDF graph* is a finite set of RDF triples.

Patterns Assume furthermore an infinite countable set V of *variables*, disjoint from U. It is a SPARQL convention to prefix each variable with a question mark. *Patterns* are now 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* or *basic triple pattern* in SPARQL 1.1).
- If P_1 and P_2 are patterns, then so are the following: P_1 UNION P_2, P_1 AND P_2, and P_1 MINUS P_2.
- 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* is a conjunction of *atomic constraints*; an atomic constraint can have one of the three following forms:
  - *bound:* $\mathrm{bound}(?x)$ with $?x \in V$;
  - *equality:* $?x = ?y$ with $?x, ?y \in V$;
  - *constant equality:* $?x = c$ with $?x \in V$ and $c \in I \cup L$.

Some clarification is in order here. Compared to the usual presentation of SPARQL patterns [38, 39], we leave out three features and we add one feature. On the one hand, we leave out the OPTIONAL operator; bound constraints; and negation in filter conditions.[1] On the other hand, we add the MINUS operator. This simplification is permitted, since MINUS can actually be expressed in terms of AND, OPTIONAL, and FILTER with negated bound constraints in filter conditions [23, Proposition 3.7], [21]. Since the only need we will have for the features we leave out is exactly the MINUS operator, we find it cleaner to just add the latter. Moreover, MINUS has been added to SPARQL 1.1 as well, so our formalization matches current practice.[2]

Semantics The semantics of patterns is defined in terms of sets of so-called *mappings*, which are simply total functions $\mu\colon S \to U$ on some finite set S of variables. We denote the domain S of μ by $\mathrm{dom}(\mu)$.

Now given an RDF graph G and a pattern P, we define the semantics of P on G, denoted by $[\![P]\!]_G$, as a set of mappings, in the following manner.

- If P is a triple pattern (v_1, v_2, v_2), then

$$[\![P]\!]_G := \{\mu\colon \{v_1, v_2, v_3\} \cap V \to U \mid (\mu(v_1), \mu(v_2), \mu(v_3)) \in G\}.$$

  Here, for any mapping μ and any constant $c \in I \cup L$, we agree that $\mu(c)$ equals c itself. In other words, mappings are extended to constants according to the identity mapping.
- If P is of the form P_1 UNION P_2, then $[\![P]\!]_G := [\![P_1]\!]_G \cup [\![P_2]\!]_G$.
- If P is of the form P_1 AND P_2, then $[\![P]\!]_G := [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$, where, for any two sets of mappings Ω_1 and Ω_2, we define

$$\Omega_1 \bowtie \Omega = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1,\ \mu_2 \in \Omega_2,\ \mu_1 \sim \mu_2\}.$$

  Here, two mappings μ_1 and μ_2 are called *compatible*, denoted by $\mu_1 \sim \mu_2$, if they agree on the intersection of their domains, i.e., if for every variable $?x \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$, we have $\mu_1(?x) = \mu_2(?x)$. Note that when μ_1 and μ_2 are compatible, their union $\mu_1 \cup \mu_2$ is a well-defined mapping; this property is used in the formal definition above.
- If P is of the form P_1 MINUS P_2, then $[\![P]\!]_G := [\![P_1]\!]_G \setminus [\![P_2]\!]_G$, where, for any two sets of mappings Ω_1 and Ω_2, we define

$$\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \neg\exists \mu_2 \in \Omega_2 : \mu_1 \sim \mu_2\}.$$

- Finally, if P is of the form P_1 FILTER C, then $[\![P]\!]_G := \{\mu \in [\![P_1]\!]_G \mid \mu(C) = \mathit{true}\}$.
  Here, for any mapping μ and constraint C, the evaluation of C on μ, denoted by $\mu(C)$, is defined by taking the boolean conjunction of the evaluations of the atomic constraints in C.
  For a bound constraint $\mathrm{bound}(?x)$, we define $\mu(\mathrm{bound}(?x))$ to be true if $?x \in \mathrm{dom}(\mu)$, and false otherwise.
  For an equality constraint $?x = ?y$, we define $\mu(?x = ?y)$ to be true if $?x, ?y \in \mathrm{dom}(\mu)$ and $\mu(?x) = \mu(?y)$, and false otherwise. Similarly, for a constant-equality constraint $?x = c$, we define $\mu(?x = c)$ to be true if $?x \in \mathrm{dom}(\mu)$ and $\mu(?x) = c$, and false otherwise.

Queries A *basic SPARQL query* is an expression of the form $\mathrm{SELECT}_S\, P$ where S is a finite set of variables and P is a pattern. Semantically, given an RDF graph G, we define $[\![\mathrm{SELECT}_S\, P]\!]_G = \{\mu|_{\mathrm{dom}(\mu)\cap S} \mid \mu \in [\![P]\!]_G\}$, where we use the common notation $f|_X$ for the restriction of a function f to a subset X of its domain.

3. NESTED REGULAR EXPRESSIONS WITH NEGATION

In this section, following the definition of nested regular expressions (nre) given by Pérez et al. [18], we introduce the extension of nre's with two kinds of negation, namely, set difference '$-$' and complementation '^{c}'.

Nested regular expressions with negation (nre$^\neg$) are defined by the following formal syntax:

$$e \;:=\; \mathit{axis} \mid \mathit{axis} :: c \mid \mathit{axis} :: [e] \mid e \circ e \mid e \cup e \mid e^* \mid e - e \mid e^c,$$

[1] We also leave out disjunction in filter conditions, but that can easily be expressed in terms of UNION.

[2] In SPARQL 1.1, the semantics we use here for MINUS is actually called DIFF.

where $a \in I \cup L$ and $axis \in \{self,\ next,\ next^{-1},\ edge,\ edge^{-1},\ node,\ node^{-1}\}$.

Given an RDF graph G, the evaluation of e on G, denoted by $[\![e]\!]_G$, is a binary relation defined in the following way. Below, let $axis \in \{next, edge, node\}$ and $adom(G)$ denotes the *active domain* of G, i.e., the set of all elements from U occurring in G. The operator * denotes reflexive-transitive closure.

$$
\begin{array}{rcl}
[\![self]\!]_G & = & \{(s,s) \mid s \in adom(G)\};\\
[\![self :: c]\!]_G & = & [\![self]\!]_G \cap \{(c,c)\};\\
[\![next]\!]_G & = & \{(s,t) \mid \exists u : (s,u,t) \in G\};\\
[\![next :: c]\!]_G & = & \{(s,t) \mid (s,c,t) \in G\};\\
[\![edge]\!]_G & = & \{(s,t) \mid \exists u : (s,t,u) \in G\};\\
[\![edge :: c]\!]_G & = & \{(s,t) \mid (s,t,c) \in G\};\\
[\![node]\!]_G & = & \{(s,t) \mid \exists u : (u,s,t) \in G\};\\
[\![node :: c]\!]_G & = & \{(s,t) \mid (c,s,t) \in G\};\\
[\![axis^{-1}]\!]_G & = & \{(s,t) \mid (t,s) \in [\![axis]\!]_G\};\\
[\![axis^{-1} :: c]\!]_G & = & \{(s,t) \mid (t,s) \in [\![axis :: c]\!]_G\};\\
[\![e_1 \cup e_2]\!]_G & = & [\![e_1]\!]_G \cup [\![e_2]\!]_G;\\
[\![e_1 \circ e_2]\!]_G & = & \{(s,t) \mid \exists u : (s,u) \in [\![e_1]\!]_G \\
 & & \text{and } (u,t) \in [\![e_2]\!]_G\};\\
[\![e_1 - e_2]\!]_G & = & \{(s,t) \in [\![e_1]\!]_G \mid (s,t) \notin [\![e_2]\!]_G\};\\
[\![e^c]\!]_G & = & \{(s,t) \in adom(G) \times adom(G) \mid \\
 & & (s,t) \notin [\![e]\!]_G\};\\
[\![e^*]\!]_G & = & [\![e]\!]_G{}^*\\
[\![self :: [e]]\!]_G & = & \{(s,s) \mid \exists u : (s,u) \in [\![e]\!]_G\};\\
[\![axis :: [e]]\!]_G & = & \{(s,t) \mid \exists u,v : (s,t) \in [\![axis :: u]\!]_G \\
 & & \text{and } (u,v) \in [\![e]\!]_G\};\\
[\![axis^{-1} :: [e]]\!]_G & = & \{(s,t) \mid (t,s) \in [\![axis :: [e]]\!]_G\}.
\end{array}
$$

REMARK 1. 1. Set difference '$-$' is actually redundant in the above since $e_1 - e_2$ can be expressed using complementation as $(e_1^c \cup e_2)^c$. We will see in Section 5 that the converse does not hold in general. However, if we allow the *universal expression all*, defined by $[\![all]\!]_G = adom(G) \times adom(G)$, then we conversely have $e^c = all - e$.
2. Intersection is a derived operator in the above, given that $e_1 \cap e_2$ can be expressed as $e_1 - (e_1 - e_2)$.
3. Complementation and set difference are much more powerful than the so-called "negated property sets" in SPARQL 1.1 property paths, which are equivalent to nre's of the form $next - (next :: a \cup \cdots \cup next :: b)$ or $next^{-1} - (next^{-1} :: a \cup \cdots \cup next^{-1} :: b)$, for a finite list $a, \ldots, b$ of constants.

□

Similarly to nSPARQL [18], we can consider *extended patterns*, which extend the definition of patterns by also allowing *nre$^\neg$ triples*: expressions of the form $(?x, e, ?y)$ with $?x, ?y$ variables and e an nre$^\neg$. Given an RDF graph G, their semantics is defined by

$$
[\![(?x, e, ?y)]\!]_G = \{\mu\colon \{?x, ?y\} \to U \mid (\mu(?x), \mu(?y)) \in [\![e]\!]_G\}.
$$

Extending the basic SPARQL queries defined in Section 2, we can then define an *nSPARQL$^\neg$ query* as an expression of the form $\mathrm{SELECT}_S\, P$, where now P can be an extended pattern.

EXAMPLE 1. Consider the toy example RDF graph $G_{sn} = \{t_1, \ldots, t_5\}$ storing information about an academic social network:

$$
\begin{array}{rcl}
t_1 & = & (\textit{C.Christian}, \textit{advisee}, \textit{G.Gottlob});\\
t_2 & = & (\textit{G.Gottlob}, \textit{advisee}, \textit{T.Eiter});\\
t_3 & = & (\textit{T.Eiter}, \textit{advisee}, \textit{A.Polleres});\\
t_4 & = & (\textit{G.Gottlob}, \textit{cowork}, \textit{A.Polleres});\\
t_5 & = & (\textit{advisee}, \textit{subPropertyOf}, \textit{cowork}).
\end{array}
$$

We give five navigational queries and for each query an nre$^\neg$ that expresses it. Each query returns all pairs (X, Y) such that some property holds:

Q1 Y advises X: $e_1 = next$::*advisee*. We have

$$
[\![e_1]\!]_{G_{sn}} = \{(\textit{C.Christian}, \textit{G.Gottlob}), (\textit{G.Gottlob}, \textit{T.Eiter}), (\textit{T.Eiter}, \textit{A.Polleres})\}.
$$

Q2 Y advises someone who advises X:

$$
e_2 = (next :: \textit{advisee}) \circ (next :: \textit{advisee}).
$$

We have

$$
[\![e_2]\!]_{G_{sn}} = \{(\textit{C.Christian}, \textit{T.Eiter}), (\textit{G.Gottlob}, \textit{A.Polleres})\}.
$$

Q3 X works with Y, considered as a reflexive and symmetric relation, and including properties that are related to the cowork property (in this simple example, the advisee property is related to the cowork property via the subPropertyOf property): $e_3 = e \cup e^{-1}$ where e is the expression

$$
next :: \textit{cowork} \cup next :: [edge :: \textit{cowork}].
$$

Then $[\![e_3]\!]_{G_{sn}}$ equals

$$
\begin{array}{c}
\{(\textit{C.Christian}, \textit{C.Christian}), (\textit{G.Gottlob}, \textit{G.Gottlob}),\\
(\textit{T.Eiter}, \textit{T.Eiter}), (\textit{A.Polleres}, \textit{A.Polleres}),\\
(\textit{C.Christian}, \textit{G.Gottlob}), (\textit{G.Gottlob}, \textit{T.Eiter}),\\
(\textit{T.Eiter}, \textit{A.Polleres}), (\textit{G.Gottlob}, \textit{A.Polleres}),\\
(\textit{G.Gottlob}, \textit{C.Christian}), (\textit{T.Eiter}, \textit{G.Gottlob}),\\
(\textit{A.Polleres}, \textit{T.Eiter}), (\textit{A.Polleres}, \textit{G.Gottlob})\}.
\end{array}
$$

Q4 Y does not work with X: $e_4 = e_3^c$. Then $[\![e_4]\!]_{G_{sn}}$ equals

$$
\begin{array}{c}
\{(\textit{C.Christian}, \textit{T.Eiter}), (\textit{C.Christian}, \textit{A.Polleres}),\\
(\textit{G.Gottlob}, \textit{A.Polleres}), (\textit{T.Eiter}, \textit{C.Christian})\\
(\textit{A.Polleres}, \textit{C.Christian}), (\textit{A.Polleres}, \textit{G.Gottlob})\}.
\end{array}
$$

Q5 Y advises someone who advises X, but Y does not work with X in the sense of **Q3**: $e_5 = e_2 - e_3$. We have $[\![e_5]\!]_{G_{sn}} = \{(\textit{C.Christian}, \textit{T.Eiter})\}$.

Q6 Y is an *advise*-ancestor of X: $e_6 = (next :: advise)^*$. Then $[\![e_6]\!]_{G_{sn}}$ equals

$$\{(\textit{C.Christian}, \textit{C.Christian}), (\textit{G.Gottlob}, \textit{G.Gottlob}), (\textit{T.Eiter}, \textit{T.Eiter}), (\textit{A.Polleres}, \textit{A.Polleres}), (\textit{C.Christian}, \textit{G.Gottlob}), (\textit{C.Christian}, \textit{T.Eiter}), (\textit{C.Christian}, \textit{A.Polleres}), (\textit{G.Gottlob}, \textit{T.Eiter}), (\textit{G.Gottlob}, \textit{A.Polleres}), (\textit{T.Eiter}, \textit{A.Polleres})\}.$$

4. FROM nSPARQL$^\neg$ TO BASIC SPARQL

As argued in the Introduction, a natural level of expressibility is formed by the fragment of nre$^\neg$ where the Kleene star operator '$*$' is not used. We call these *star-free* (sfnre$^\neg$). Accordingly, an *sfnSPARQL$^\neg$ query* is an nSPARQL$^\neg$ query where the Kleene star operator is not used in extended patterns.

In this section, we introduce *navigation patterns* and show how sfnre$^\neg$'s can be translated into them. This provides an elegant proof of the result that every sfnSPARQL$^\neg$ query can already be expressed as a basic SPARQL query.

A navigation pattern is a triple $(P, ?x, ?y)$ where P is a pattern (a basic pattern, not an extended pattern) and $?x$ and $?y$ are variables occurring in P. We call a navigation pattern *sound* if for every RDF graph G, and every $\mu \in [\![P]\!]_G$, we always have $?x$ and $?y$ in $\mathrm{dom}(\mu)$. Sound navigation patterns can be used to express binary relations on RDF graphs, which will allow us to compare navigation patterns with sfnre$^\neg$'s. Formally, given any RDF graph G we define a binary relation on every RDF graph G simply by $[\![(P, ?x, ?y)]\!]_G = \{(\mu(?x), \mu(?y)) \mid \mu \in [\![P]\!]_G\}$.

A first question, however, is how soundness can be syntactically guaranteed. Thereto, we will introduce *safe* navigation patterns. For this, we first introduce the following notion.

DEFINITION 4.1. *The* mapping schema *of a pattern P is a set $MS(P)$ of sets of variables, defined inductively as follows:*

- *If P is a triple pattern (v_1, v_2, v_2) then $MS(P) := \{\{v_1, v_2, v_3\} \cap V\}$.*
- *If P is of the form P_1 UNION P_2, then $MS(P) := MS(P_1) \cup MS(P_2)$.*
- *If P is of the form P_1 AND P_2, then*

$$MS(P) := \{S_1 \cup S_2 \mid S_1 \in MS(P_1),\ S_2 \in MS(P_2)\}.$$

- *Finally if P is of the form P_1 MINUS P_2 or of the form P_1 FILTER C, then $MS(P) := MS(P_1)$.*

The relevant property of mapping schemas is that they cover all possible domains of mappings that can occur in the result of the pattern.

PROPOSITION 4.1. *For any RDF graph G, $\mu \in [\![P]\!]_G$ implies $\mathrm{dom}(\mu) \in MS(P)$.*

Proof. By induction on the structure of P. If P is a triple pattern (v_1, v_2, v_3) then for any graph G and for any $\mu \in [\![P]\!]_G$, we have $\mathrm{dom}(\mu) = \{v_1, v_2, v_3\} \cap V$ which belongs to $MS(P)$ by definition.

If P is of the form P_1 UNION P_2, P_1 MINUS P_2, or P_1 FILTER C, the claim readily follows by induction.

If P is of the form P_1 AND P_2, let $\mu \in [\![P]\!]_G$. There exist $\mu_1 \in [\![P_1]\!]_G$ and $\mu_2 \in [\![P_2]\!]_G$ such that $\mu = \mu_1 \cup \mu_2$ and $\mu_1 \sim \mu_2$. By induction, $\mathrm{dom}(\mu_1) \in MS(P_1)$ and $\mathrm{dom}(\mu_2) \in MS(P_2)$. Then $\mathrm{dom}(\mu) = \mathrm{dom}(\mu_1) \cup \mathrm{dom}(\mu_2) \in MS(P_1 \text{ AND } P_2)$ by definition. □

Hence, we can syntactically guarantee soundness by the following definition and proposition.

DEFINITION 4.2. *A navigation pattern $(P, ?x, ?y)$ is called* safe *if $?x$ and $?y$ belong to every set in $MS(P)$.*

PROPOSITION 4.2. *Every safe navigation pattern is sound.*

Proof. Let $(P, ?x, ?y)$ be a navigation pattern. If $(P, ?x, ?y)$ is safe then, by definition, $\{?x, ?y\} \subseteq S$ for all $S \in MS(P)$. By Proposition 4.1, for any graph G and for any $\mu \in [\![P]\!]_G$, $\mathrm{dom}(\mu) \in MS(P)$. Then $\{?x, ?y\} \subseteq \mathrm{dom}(\mu)$. Therefore, $(P, ?x, ?y)$ is sound. □

The converse direction does not hold. For example, any unsatisfiable pattern, safe or not, trivially yields a sound navigation pattern. Satisfiability of patterns is undecidable, and soundness can be shown to be undecidable as well. This does not concern us here, however, since we will only work with safe navigation patterns.

The main result of this section is that sfnre$^\neg$ expressions can be translated into safe navigation patterns. Before giving the formal theorem and proof, we give some illustrative examples.

EXAMPLE 2. Recall the sfnre$^\neg$'s e_1, e_2, e_3 and e_5 from Example 1. We can express these by the navigation patterns $(P_i, ?x, ?y)$ as follows:

- $P_1 = (?x, \textit{advisee}, ?y)$.
- $P_2 = (?x, \textit{advisee}, ?z)$ AND $(?z, \textit{advisee}, ?y)$.
- $P_3 = P_{31}$ UNION P_{32} where

$$P_{31} = (?x, \textit{cowork}, ?y) \text{ UNION } ((?x, ?z, ?y) \text{ AND } (?z, ?y', \textit{cowork}))$$

  and P_{32} is like P_{31} but with $?x$ and $?y$ swapped.
- $P_5 = P_2$ MINUS P_3.

□

In the proof of the theorem below we use $vars(P)$ to denote the set of variables that occur in the pattern P.

THEOREM 4.1. *For every sfnre$^\neg$ e there exists a safe navigation pattern $(P, ?x, ?y)$ such that for every RDF graph G, we have $[\![e]\!]_G = [\![(P, ?x, ?y)]\!]_G$.*

Proof. By induction on the structure of e. If e is *self* we set $P = (adom_{?x} \text{ AND } adom_{?y})$ FILTER $?x = ?y$, where we use the shorthand $adom_v$, for any variable v, to denote

$$(v, v', v'') \text{ UNION } (v', v, v'') \text{ UNION } (v', v'', v)$$

with v' and v'' fresh auxiliary variables. As a variant, if e is $self :: c$ then we use $P' = P \,\text{FILTER}\, ?x = c$ with P as above.

If e is $next$, $edge$, or $node$, we set $P = (?x, ?z, ?y)$, $(?x, ?y, ?z)$, or $(?z, ?x, ?y)$ respectively, where $?z$ is a fresh auxiliary variable. Similarly, if e is $next :: c$, $edge :: c$, or $node :: c$, we set $P = (?x, c, ?y)$, $(?x, ?y, c)$, or $(c, ?x, ?y)$ respectively. When e is of the form $axis^{-1}$ or $axis^{-1} :: c$, with $axis \in \{next, edge, node\}$, we swap $?x$ and $?y$ in the above.

If e is of the form $e_1 \cup e_2$ then by induction we have a navigation pattern $(P_1, ?x_1, ?y_1)$ and $(P_2, ?x_2, ?y_2)$ for e_1 and e_2 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 $?x$ and $?y$.

Then $(P_1' \text{ UNION } P_2', ?x, ?y)$ is a navigation pattern for e.

The case where e is of the form $e_1 - e_2$ is similar, using MINUS instead of UNION.

If e is of the form e^c then by induction, we have navigation patterns $(P, ?x, ?y)$ for e. Then $((adom_{?x} \text{ AND } adom_{?y}) \text{ MINUS } P, ?x, ?y)$ is a navigation pattern for e^c.

If e is of the form $e_1 \circ e_2$, again let $(P_i, ?x_i, ?y_i)$ be a navigation pattern for e_i, for $i = 1, 2$. Now let P_1' and P_2' be obtained from P_1 and P_2 by renaming the variables so that

- $?x_1$ is renamed to $?x$;
- $?y_1$ is renamed to $?z$, with $?z$ distinct from both $?x$ and $?y$;
- $?x_2$ is renamed to the same $?z$;
- $?y_2$ is renamed to $?y$; and
- P_1' and P_2' have no common variables other than $?x$, $?y$ and $?z$.

Then $(P_1' \text{ AND } P_2', ?x, ?y)$ is a navigation pattern for e.

If e is of the form $axis :: [e']$, with $axis \in \{next, edge, node\}$, by induction we have a navigation pattern $(P', ?x', ?y')$ for e'. We also have a navigation pattern $(P, ?x, ?y)$ for $axis$ as described above. Recall that P involves an auxiliary variable $?z$. We obtain P'' by renaming variables in P' so that $?x'$ is renamed to $?z$, and so that P and P'' have no common variables other than $?z$. We can then use $(P \text{ AND } P'', ?x, ?y)$ as a navigation pattern for e. When e is of the $axis^{-1} :: [e']$, with $axis \in \{next, edge, node\}$, we swap $?x$ and $?y$.

Finally, if e is of the form $self :: [e']$, by induction we have a navigation pattern $(P', ?x', ?y')$ for e'. We can use $(((adom_{?x} \text{ AND } adom_{?y}) \text{ FILTER } ?x = ?y) \text{ AND } P'', ?x, ?y)$ as a navigation pattern for $self :: [e]$ where P'' is obtained from P' by renaming the variables so that

- $?x'$ is renamed to $?x$;
- $?y$ does not occur in P''.

□

As a corollary to the above theorem we obtain:

COROLLARY 4.1. *Every sfnSPARQL$^\neg$ query is already expressible as a basic SPARQL query.*

Proof. An sfnre$^\neg$ triple pattern $(?x, e, ?y)$ is replaced by the navigation pattern for e given by Theorem 4.1. This pattern uses a lot of auxiliary variables; these are chosen freshly for each occurrence of an sfnre$^\neg$ triple pattern, so that the auxiliary variables do not interact in applications of AND, UNION, and MINUS occurring in the query. In the final projection performed by the SELECT query, all auxiliary variables are projected away. □

4.1. Tarski-SPARQL

To conclude this section, we note that the translation provided in the proof of Theorem 4.1 is effective (in fact, linear), and moreover, the image of the translation can be captured syntactically. In this way we obtain a syntactic fragment of the navigation patterns that has exactly the same expressive power as sfnre$^\neg$'s. In the spirit of the Introduction, we will call these the *Tarski-navigation patterns*.

Formally, the Tarski-navigation patterns (Tnp's) are inductively defined as follows:

1. For any variables $?x$ and $?y$, the triple $(P, ?x, ?y)$ is a Tnp, where P equals $(adom_{?x} \text{ AND } adom_{?y}) \,\text{FILTER}\, ?x = ?y$.
2. For any three distinct variables $?x$, $?y$ and $?z$, and constant c, the triple $(P, ?x, ?y)$ is a Tnp for all the following possibilities for P: $(?x, ?z, ?y)$; $(?x, ?y, ?z)$; $(?z, ?x, ?y)$; $(?x, c, ?y)$; $(?x, ?y, c)$; and $(c, ?x, ?y)$.
3. If $(P_1, ?x, ?y)$ and $(P_2, ?x, ?y)$ are Tnp's so that P_1 and P_2 have no common variables other than $?x$ and $?y$, then also $(P_1 \text{ UNION } P_2, ?x, ?y)$ and $(P_1 \text{ MINUS } P_2, ?x, ?y)$ are Tnp's.
4. If $(P_1, ?x, ?z)$ and $(P_2, ?z, ?y)$ are Tnp's so that P_1 and P_2 have no common variables other than $?x$, $?y$ and $?z$, then also $(P_1 \text{ AND } P_2, ?x, ?y)$ is a Tnp.
5. If $(P', ?z, ?u)$ is a Tnp, then also $(P \text{ AND } P', ?x, ?y)$ is a Tnp for all the following possibilities for P: $(?x, ?z, ?y)$; $(?x, ?y, ?z)$; and $(?z, ?x, ?y)$.
6. If $(P, ?x, ?y)$ is a Tnp and $?z$ is not a variable of P. then also $(((adom_{?x} \text{ AND } adom_{?z}) \,\text{FILTER}\, ?x = ?z) \text{ AND } P, ?x, ?z)$ is a Tnp.
7. If $(P, ?x, ?y)$ is a Tnp, then also $(adom_{?x} \text{ AND } adom_{?y}) \text{ MINUS } P, ?x, ?y)$ is a Tnp.
8. If $(P, ?x, ?y)$ is a Tnp, then so is $(P, ?y, ?x)$.

The language of Tnp's is equivalent to that of sfnre$^\neg$'s in the following sense. A *binary-relation query* is a mapping from RDF graphs to binary relations. Note that the semantics of an sfnre$^\neg$, as well as the semantics of a navigation pattern, is a binary-relation query.

THEOREM 4.2. *A binary-relation query is expressible as an sfnre$^\neg$ if and only if it is expressible as a Tnp.*

Proof. The only-if direction can be verified by observing that the proof of Theorem 4.1 uses only Tnp's. It remains to show that for every Tnp $(P, ?x, ?y)$, there exists some sfnr$^\neg$ e such that for any graph G, $[\![(P, ?x, ?y)]\!]_G = [\![e]\!]_G$ by induction on the structure of P.

We follow the syntactic cases in the above definition of

Tnp's.

1. If P is of the form $(adom_{?x}$ AND $adom_{?y})$ FILTER $?x$ $=?y$ then $[\![(P, ?x, ?y)]\!]_G = [\![self]\!]_G$.
2. If P is of the form $(?x, ?z, ?y)$, $(?x, ?y, ?z)$, $(?z, ?x, ?y)$, $(?x, c, ?y)$, $(?x, ?y, c)$, or $(c, ?x, ?y)$, respectively, then $[\![(P, ?x, ?y)]\!]_G$ equals $[\![next]\!]_G$, $[\![edge]\!]_G$, $[\![node]\!]_G$, $[\![next :: c]\!]_G$, $[\![edge :: c]\!]_G$, and $[\![node :: c]\!]_G$, respectively.
3. If P is of the form P_1 UNION P_2, we have sfnre$^\neg$'s e_1 and e_2 for $(P_1, ?x, ?y)$ and $(P_2, ?x, ?y)$ respectively. We then take $e_1 \cup e_2$ for $(P, ?x, ?y)$. If P is of the form P_1 MINUS P_2 we take $e_1 - e_2$.
4. If P is of the form P_1 AND P_2, for Tnp's $(P_1, ?x, ?z)$ and $(P_2, ?z, ?y)$ are Tnp's, we let $e = e_1 \circ e_2$.
5. If P is of the form $(?x, ?z, ?y)$ AND P', for Tnp $(P', ?z, ?w)$, we let $e = next :: [e']$ where $[\![(P', ?z, ?w)]\!]_G = [\![e']\!]_G$. Analogously, if P is of the form $(?x, ?y, ?z)$ AND P' or $(?z, ?x, ?y)$ AND P', respectively, we take $e = edge :: [e']$ or $e = node :: [e']$ respectively.
6. If P is of the form $((adom_{?x}$ AND $adom_{?y})$ FILTER $?x$ $=?y)$ AND P', for Tnp $(P', ?y, ?z)$, we let let $e = self :: [e']$.
7. If P is of the form $(adom_{?x}$ AND $adom_{?y})$ MINUS P', for Tnp $(P', ?x, ?y)$, we let $e = (e')^c$.
8. Finally, if e is an sfnre$^\neg$ for the Tnp $(P, ?x, ?y)$, then e^{-1} is an sfnre$^\neg$ for the Tnp $(P, ?y, ?x)$, where e^{-1} is defined as follows:
   - $self^{-1} = self$;
   - $(self :: c)^{-1} = self :: c$;
   - $(self :: [e])^{-1} = self :: [e]$;
   - $(e_1 \cup e_2)^{-1} = e_1^{-1} \cup e_2^{-1}$;
   - $(e_1 - e_2)^{-1} = e_1^{-1} - e_2^{-1}$;
   - $(e_1 \circ e_2)^{-1} = e_2^{-1} \circ e_1^{-1}$;
   - $(e^c)^{-1} = (e^{-1})^c$;
   - $(e^*)^{-1} = (e^{-1})^*$;
   - $(axis)^{-1} = axis^{-1}$;
   - $(axis :: c)^{-1} = axis^{-1} :: c$.
   - $(axis :: [e])^{-1} = axis^{-1} :: [e]$;

   with $axis \in \{next, edge, node\}$.

□

We thus obtain a clean syntactic fragment of basic SPARQL queries, which may be called *Tarski-SPARQL*, that precisely delineates the sfnre$^\neg$'s. The Tarski-SPARQL queries are precisely all queries of the form SELECT$_{?x,?y}$ P where $(P, ?x, ?y)$ is a Tarski-navigation pattern.

5. THE EXPRESSIVITY OF NESTED REGULAR EXPRESSIONS WITH NEGATION

In this section, we investigate and highlight various facets of the expressivity of nre$^\neg$'s.

5.1. Set difference versus complementation

We first consider a fragment called *nested regular expression with set difference* (nre$^-$), that is, complement-free nested regular expressions.

We point out that nre$^-$ are, in a sense, more intuitive than nre$^\neg$ in the context of navigational querying. Intuitively one expects of a navigational query to an RDF graph G to return only pairs (a, b) of elements that are "connected" in G. We can formalize this intuition as follows. Let G be an RDF graph and let $a, b \in adom(G)$. The relation of *a and b being connected in G* is the reflexive, symmetric, and transitive closure of the set of all pairs (a, b) that occur jointly in an RDF triple in G. By default, for any $a \in adom(G)$, a always connects to itself.

We can confirm that nre$^-$'s as defined in this paper are indeed "navigational" in this sense:

PROPOSITION 5.1. *For every nre$^-$ e, for every RDF graph G, and for every $(a, b) \in [\![e]\!]_G$, a and b are connected in G.*

Proof. By induction on the structure of e. For any graph G, if e is of the form $self$ then any $(a, a) \in [\![self]\!]_G$ satisfies $a = b$ and $a \in adom(G)$. By definition a connects to itself in G. Similarly, if e is of the form $self : c$ then the only element of $[\![self :: c]\!]_G$ is (c, c) with $c \in adom(G)$.

If e is of the form $axis$, or $axis :: c$, or $axis :: [e']$, where $axis$ may be inverted, any pair $(a, b) \in [\![next]\!]_G$ occurs jointly in an RDF triple by definition of the semantics of nre's.

If $(a, b) \in [\![e_1 \circ e_2]\!]_G$ then there exists c such that $(a, c) \in [\![e_1]\!]_G$ and $(c, b) \in [\![e_2]\!]_G$. By induction, a and c are connected, and c and b are connected, so by transitivity a and b are connected.

If e is of the form $e_1 \cup e_2$ the claim follows immediately by induction.

If $(a, b) \in [\![e_1 - e_2]\!]_G$ then in particular $(a, b) \in [\![e_1]\!]_G$ so again the claim follows by induction.

Finally if $(a, b) \in [\![e'^*]\!]_G$ then by induction (a, b) belongs to the reflexive-transitive closure of the connectedness relation. Since the connectedness relation is transitively closed, a and b are themselves connected.

□

When adding the complement operator, this property is evidently lost. For instance, let $G = \{(a, b, c), (a', b', c')\}$ be a graph with $\{a, b, c\} \cap \{a', b', c'\} = \emptyset$. Then $(a, a') \in [\![(next)^c]\!]_G$ but a and a' are not connected in G. This shows that nre$^-$ is a proper fragment of nre$^\neg$.

5.2. Expressing the residuals

The above notwithstanding, star-free nre$^\neg$'s do allow the expression of interesting derived operators, such as the *residuals*.

Formally, the *right residual* $(e_1 \swarrow e_2)$ and the *left residual* $(e_1 \searrow e_2)$ are defined as follows:

$$
\begin{aligned}
[\![e_1 \swarrow e_2]\!]_G &= \{(s, t) \mid \forall v \in adom(G) : \\
&\qquad (t, v) \in [\![e_2]\!]_G \rightarrow (s, v) \in [\![e_1]\!]_G\}; \\
[\![e_1 \searrow e_2]\!]_G &= \{(s, t) \mid \forall v \in adom(G) : \\
&\qquad (v, s) \in [\![e_1]\!]_G \rightarrow (v, t) \in [\![e_2]\!]_G\}.
\end{aligned}
$$

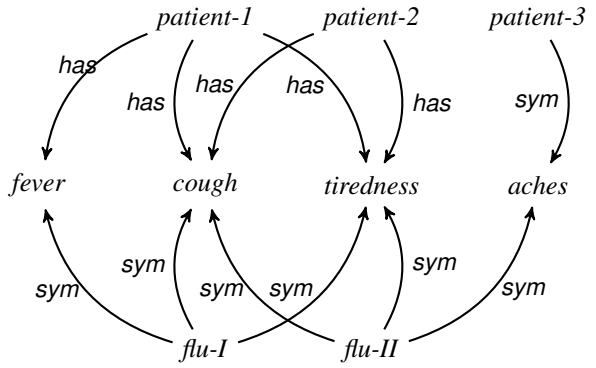

FIGURE 1. RDF graph about patients, diseases, and symptoms.

A classical example is the following.

EXAMPLE 3. Consider RDF graphs about patients, diseases, and symptoms. An example is the graph G_{pd} shown in Figure 1. There are two relations: a relation symptom (*sym*) connecting diseases to symptoms, and a relation diagnosis (*has*) connecting patient to symptoms. Consider the query Q asking for all pairs (p, d) so that patient p has all the symptoms of disease d. For example, $Q(G_{pd}) = \{(\textit{patient-1}, \textit{flu-I})\}$. Query **Q** can be expressed by $next :: \textit{has} \swarrow next :: \textit{sym}$.

It is well known that the residuals are expressible in the calculus of relations [15]: we have $e_1 \swarrow e_2 \equiv (e_1^c \circ e_2^{-1})^c$ and $e_1 \searrow e_2 \equiv (e_1^{-1} \circ e_2^c)^c$. These expressions apply inverse to entire subexpressions, but inversion can always be pushed down so that it is only applied to atomic expressions, such as in the way nre$^\neg$'s are defined.

5.3. Three-variable first-order logic

Another interesting question concerns the relationship to FO^3: the fragment of first-order logic using only three distinct variables. We can use first-order logic formulas to express binary-relation queries over RDF graphs by viewing an RDF graph G as the interpretation of a ternary relation symbol T. Formally, we associate to G a structure I_G over T whose domain equals $adom(G)$. The binary-relation query Q expressed by a formula $\varphi(x, y)$ with two free variables x and y is then simply defined by $Q(G) = \{(a, b) \mid a \in adom(G)$ and $b \in adom(G)$ and $I_G \models \varphi[a, b]\}$.

We have:

PROPOSITION 5.2. *For every sfnre$^\neg$ e, the binary-relation query $[\![e]\!]$ is expressible in FO^3.*

Proof. Clear from the formal semantics of sfnre's as given in Section 3, except perhaps for node tests (expressions of the form $axis :: [e]$). By induction, we have an FO^3 formula $\varphi(x, y)$ expressing $[\![e]\!]$. Formula φ may use a third variable z in addition to its free variables x and y. Let φ' be the formula obtained from φ by replacing x by z, y by x, and z by y. Then $[\![next :: [e]]\!]$ is expressed by the FO^3 query

$$\{(x, y) \mid \exists z(T(x, z, y) \wedge \exists x \exists y\, \varphi')\}.$$

The other axes are handled similarly. □

For the standard calculus of relations over binary relations, a converse to Proposition 5.2 is known: it can express *exactly* the binary-relation queries over binary relational structures that are expressible in FO^3 [14, 32]. It is natural to wonder about an analogous result for binary-relation queries over RDF graphs. It is tempting to conjecture that the converse to Proposition 5.2 holds, that is, that every binary-relation query over RDF graphs expressible in FO^3 is expressible by an sfnre$^\neg$. The answer is negative, as the following proposition shows a very intuitive FO^3-query, asking for symmetric relationships, to be not expressible. Note that the result actually holds for nre$^\neg$, i.e., transitive closure is allowed.

PROPOSITION 5.3. *The binary-relation query $\varphi = \{(x, y) \mid \exists z : (T(x, z, y) \wedge T(y, z, x))\}$ is not expressible by any nre$^\neg$.*

Proof. Suppose, for the sake of contradiction, that φ would be expressible by some nre$^\neg$ e. Let G be an RDF graph built from four constants a, b, c, d that do not occur in e, such that $G = \{t_1, \ldots, t_{12}\}$ where

$$\begin{array}{lll} t_1 = (a, b, c), & t_2 = (b, c, a), & t_3 = (c, a, b), \\ t_4 = (a, d, b), & t_5 = (b, a, d), & t_6 = (d, b, a), \\ t_7 = (b, d, c), & t_8 = (c, b, d), & t_9 = (d, c, b), \\ t_{10} = (a, c, d), & t_{11} = (c, d, a), & t_{12} = (d, a, c). \end{array}$$

Let $H = G \cup \{(c, b, a)\}$.

On the one hand, $\varphi(G) = \emptyset$ but $\varphi(H) = \{(a, c), (c, a)\} \neq \emptyset$.

On the other hand, we claim that

CLAIM 1. For every nre$^\neg$ e that does not use the constants a, b, c, d, we have $[\![e]\!]_G = [\![e]\!]_H$ and $[\![e]\!]_G \in \{\emptyset, \mathcal{I}, \mathcal{D}, \mathcal{I} \cup \mathcal{D}\}$ where

- $\mathcal{I} = \{(a, a), (b, b), (c, c), (d, d)\}$;
- $\mathcal{D} = \{a, b, c, d\} \times \{a, b, c, d\} - \mathcal{I}$.

In view of $\varphi(G) = \emptyset \neq \varphi(H)$, this claim implies that φ cannot be correctly expressed by e.

We prove the claim by induction.

If e is of the form *self* then $[\![e]\!]_G = [\![e]\!]_H = \mathcal{I}$.

If e is of the form $axis$ or $axis^{-1}$, for $axis \in \{next, edge, node\}$, then one may verify that $[\![e]\!]_G = [\![e]\!]_H = \mathcal{D}$.

If e is of the form $axis :: u$ or $axis^{-1} :: u$, with $u \notin \{a, b, c, d\}$, then $[\![axis :: u]\!]_G = [\![axis :: u]\!]_H = \emptyset$.

If e is of the form $e_1 \cup e_2$, $e_1 - e_2$, $e_1 \circ e_2$, e_1^c, or e_1^*, the claim follows by induction, given that the set of the four possible relations $\{\emptyset, \mathcal{I}, \mathcal{D}, \{a, b, c, d\}^2\}$ is closed under the operations union, difference, composition, complementation, and reflexive-transitive closure.

If e is of the form $self :: [e']$ then for $J = G$ and $J = H$ alike, we have $[\![e]\!]_J = \emptyset$ if $[\![e']\!]_J = \emptyset$, and $[\![e]\!]_J = \mathcal{I}$ otherwise.

Finally, if e is of the form $axis :: [e']$ then for $J = G$ and $J = H$ alike, we can reason as follows. By induction,

there are four possibilities for $[\![e']\!]_J$. If $[\![e']\!]_J = \emptyset$ then $[\![e]\!]_J = \emptyset$. If $[\![e']\!]_J$ equals $\mathcal{I}$, $\mathcal{D}$, or $\{a, b, c, d\}^2$, then $[\![axis :: [e']]\!]_J = [\![axis]\!]_J$ which we have already verified above to be equal to $\mathcal{D}$. □

6. DISCUSSION

In connection to Proposition 5.3, we should mention the TriAL language [29], which offers in its fragment TriAL$^=$ a level of expressiveness between FO^3 and FO^4. It would be interesting to find an nre-like language exactly equivalent to FO^3 over ternary relations.

Another question could be asked in connection to Proposition 5.1, where we have seen that sfnre's enjoy a connectivity property whereas general sfnre$^\neg$ queries do not. Hence one may ask for a natural characterization of the sfnre$^\neg$ queries that do enjoy the connectivity property of Proposition 5.1. A similar question could be asked for FO^3 instead of sfnre$^\neg$. We are currently investing the above and related questions. Note that for the language nre$^\neg$ (which includes transitive closure) the question is easily answered: the set of all expressions of the form

$$e \cap (self \cup next \cup edge \cup node)^*$$

where e ranges over arbitrary nre$^\neg$ expressions, captures exactly all nre$^\neg$ queries that are connected. In the absence of transitive closure, such an easy answer is less obvious, since by the mere guarantee that an expression expresses a connected query, one does not get an immediate bound on the length of the involved paths. Of course, as a recourse one may simply give a nonstandard semantics to sfnre$^\neg$ expressions to the effect of always intersecting their result with $(self \cup next \cup edge \cup node)^*$. But we do not find this a natural characterization; the question is whether the connected sfnre$^\neg$ queries can be characterized as a syntactic fragment of the language sfnre$^\neg$ itself.

Obviously, apart from the expressivity issues that were the focus of the present paper, one may also wonder about issues related to satisfiability checking or other inference problems. Unfortunately, the situation is rather bleak in this respect. Even over structures consisting of a single binary relation, we can show that the finite satisfiability problem for expressions involving just the operators union, composition, and difference, is already undecidable [40, 41]. When removing the difference operator, satisfiability is regained, as follows from known results on ICPDL [42]. Since these results hold only for binary relation structures, it would be interesting to reconsider them in the light of ternary relations as they occur in RDF graphs. Also, the *finite* satisfiability problem remains open.

We note that there is some similarity between Tarski-SPARQL and its correspondence to sfnre$^\neg$'s, and the work on translating full SPARQL into the relational algebra for relational databases [43]. In this paper, we have been focusing on a lower level of expressibility than the full SPARQL language and have been attempting the opposite direction, translating from the algebraic sfnre$^\neg$'s to Tarski-SPARQL.

In conclusion, we hope our paper has helped to show the relevance of the classical calculus of relations in highlighting the navigational aspects of SPARQL queries. It is worth noting that Versa, one of the earliest RDF query language proposals, is much more explicitly navigational than SPARQL, which evolved from an originally lightweight language for pattern matching in graphs to a full-fledged SQL analog for RDF.

7. ACKNOWLEDGMENT

We thank the anonymous referees for their constructive comments. This work is supported by the project of Research Foundation Flanders (FWO) under grant G.0489.10N.

REFERENCES

[1] Angles, R. and Gutierrez, C. (2008) Survey of graph database models. *ACM Computing Surveys*, **40**, article 1.

[2] Abiteboul, S., Buneman, P., and Suciu, D. (2000) *Data on the Web: From relations to semistructured data and XML*. Morgan Kaufmann.

[3] Florescu, D., Levy, A., and Mendelzon, A. (1998) Database techniques for the World-Wide Web: A survey. *SIGMOD Record*, **27**, 59–74.

[4] Halevy, A., Franklin, M., and Maier, D. (2006) Principles of dataspace systems. *Proceedings 25th ACM Symposium on Principles of Database Systems*, pp. 1–9.

[5] Bizer, C., Heath, T., and Berners-Lee, T. (2009) Linked data—the story so far. *International Journal on Semantic Web and Information Systems*, **5**, 1–22.

[6] Manola, F. and Miller, E. (2004) RDF primer. W3C Recommendation.

[7] Harris, S. and Seaborne, A. (2013) SPARQL 1.1 query language. W3C Recommendation.

[8] Marx, M. and de Rijke, M. (2005) Semantic characterizations of navigational XPath. *SIGMOD Record*, **34**, 41–46.

[9] ten Cate, B. and Marx, M. (2007) Navigational XPath: Calculus and algebra. *SIGMOD Record*, **36**, 19–26.

[10] Fletcher, G., Gyssens, M., Leinders, D., Van den Bussche, J., Van Gucht, D., Vansummeren, S., and Wu, Y. (2011) Relative expressive power of navigational querying on graphs. *Proceedings 14th International Conference on Database Theory*.

[11] Libkin, L., Martens, W., and Vrgoč, D. (2013) Quering graph databases with XPath. *Proceedings 16th International Conference on Database Theory*. ACM.

[12] Angles, R., Barceló, P., and Rios, G. (2013) A practical query language for graph dbs. In Bravo, L. and Lenzerini, M. (eds.), *Proceedings 7th Alberto Mendelzon International Workshop on Foundations of Data Management*, CEUR Workshop Proceedings, **1087**.

[13] Tarski, A. (1941) On the calculus of relations. *Journal of Symbolic Logic*, **6**, 73–89.

[14] Tarski, A. and Givant, S. (1987) *A Formalization of Set Theory Without Variables*, AMS Colloquium Publications, **41**. American Mathematical Society.

[15] Pratt, V. (1992) Origins of the calculus of binary relations. *Proceedings 7th Annual IEEE Symposium on Logic in Computer Science*, pp. 248–254.

[16] Sarathy, V., Saxton, L., and Van Gucht, D. (1993) Algebraic foundation and optimization for object based query languages. *Proceedings 9th International Conference on Data Engineering*, pp. 81–90. IEEE Computer Society.

[17] Gyssens, M., Saxton, L., and Van Gucht, D. (1994) Tagging as an alternative to object creation. In Freytag, J., Maier, D., and Vossen, G. (eds.), *Query Processing For Advanced Database Systems*, chapter 8. Morgan Kaufmann.

[18] Pérez, J., Arenas, M., and Gutierrez, C. (2010) nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, **8**, 255–270.

[19] Arenas, M. and Pérez, J. (2012) Federation and navigation in SPARL 1.1. In Eiter, T. and Krennwallner, T. (eds.), *Reasoning Web: Semantic Technologies for Advanced Query Answering*, Lecture Notes in Computer Science, **7487**, pp. 78–111. Springer.

[20] Alkhateeb, F., Baget, J.-F., and Euzenat, J. (2009) Extending SPARQL with regular expression patterns (for querying RDF). *Journal of Web Semantics*, **7**, 57–73.

[21] Angles, R. and Gutierrez, C. (2008) The expressive power of SPARQL. In Sheth, A., Staab, S., et al. (eds.), *Proceedings 7th International Semantic Web Conference*, Lecture Notes in Computer Science, **5318**, pp. 114–129. Springer.

[22] Polleres, A. (2007) From SPARQL to rules (and back). In Williamson, C., Zurko, M., et al. (eds.), *Proceedings 16th World Wide Web Conference*, pp. 787–796. ACM.

[23] Arenas, M. and Pérez, J. (2011) Querying semantic web data with SPARQL. *Proceedings 30st ACM Symposium on Principles of Databases*, pp. 305–316. ACM.

[24] Alkhateeb, F. (2008) Querying RDF(S) with regular expressions. PhD thesis Université Joseph Fourier, Grenoble.

[25] Garcia-Molina, H., Ullman, J., and Widom, J. (2009) *Database Systems: The Complete Book*. Prentice Hall.

[26] Newman, M. (2003) The structure and function of complex networks. *SIAM Review*, **45**, 167–256.

[27] Watts, D. (1999) *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton University Press.

[28] Facebook Developers (2013). Facebook query language (FQL) overview. `https://developers.facebook.com/docs/technical-guides/fql/`, retrieved 5 December.

[29] Libkin, L., Reutter, J., and Vrgoč, D. (2013) TriAL for RDF: Adapting graph query languages for RDF data. *Proceedings 32nd ACM Symposium on Principles of Database Systems*, pp. 201–212. ACM.

[30] McNaughton, R. and Papert, S. (1971) *Counter-Free Automata*. MIT Press.

[31] Thomas, W. (1997) Languages, automata, and logic. In Rozenberg, G. and Salomaa, A. (eds.), *Handbook of Formal Languages*, chapter 7. Springer.

[32] Marx, M. and Venema, Y. (1997) *Multi-Dimensional Modal Logic*. Springer.

[33] Baader, F., Calvanese, D., McGuiness, D., Nardi, D., and Patel-Schneider, P. (eds.) (2003) *The Description Logic Handbook*. Cambridge University Press.

[34] Harel, D., Kozen, D., and Tiuryn, J. (2000) *Dynamic Logic*. MIT Press.

[35] Blackburn, P., van Benthem, J., and Wolter, F. (eds.) (2007) *Handbook of Modal Logic*. Elsevier.

[36] Maddux, R. (2006) *Relation Algebras*. Elsevier.

[37] Graefe, G. and Cole, R. (1995) Fast algorithms for universal quantification in large databases. *ACM Transactions on Database Systems*, **20**, 187–236.

[38] Pérez, J., Arenas, M., and Gutierrez, C. (2009) Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, **34**, article 16.

[39] Arenas, M., Pérez, J., and Gutierrez, C. (2009) On the semantics of SPARQL. In De Virgilio, R., Giunchiglia, F., and Tanca, L. (eds.), *Semantic Web Information Management—A Model-Based Perspective*, pp. 281–307. Springer.

[40] 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.

[41] Zhang, X. and Van den Bussche, J. (2014) On the satisfiability problem for SPARQL patterns. arXiv:1406.1404.

[42] Göller, S., Lohrey, M., and Lutz, C. (2009) PDL with intersection and converse: satisfiability and infinite-state model checking. *Journal of Symbolic Logic*, **74**, 279–314.

[43] Cyganiak, R. (2005) A relational algebra for SPARQL. Technical Report HPL-2005-170. HP Labs.