Knowledge Graphs: Research Directions

Aidan Hogan

IMFD; DCC, Universidad de Chile

Abstract. In these lecture notes, we provide an overview of some of the high-level research directions and open questions relating to knowledge graphs. We discuss six high-level concepts relating to knowledge graphs: data models, queries, ontologies, rules, embeddings and graph neural networks. While traditionally these concepts have been explored by different communities in the context of graphs, more recent works have begun to look at how they relate to one another, and how they can be unified. In fact, at a more foundational level, we can find some surprising relations between the different concepts. The research questions we explore mostly involve combinations of these concepts.

1 Introduction

Knowledge graphs have been gaining more and more attention in recent years, particularly in settings that involve integrating and making sense of diverse data at large scale. Much of this attention stems from the 2012 announcement of the Google Knowledge Graph [64], which was later followed by further announcements of knowledge graphs by eBay, Facebook, IBM, Microsoft [53], and many more besides [35]. These *enterprise knowledge graphs* are typically internal to a company, where they aim to serve – per the words of Noy et al. [53] – as a common substrate of knowledge within the organisation, expanding and evolving over time. Other *open knowledge graphs* – such as BabelNet [49], DBpedia [44], Freebase [14], Wikidata [67], YAGO [33] – are made available to the public.

On a more technical level, there are varying perspectives about how knowledge graphs should be formally defined (if at all) [35]. This stems from the diversity of communities and organisations that have adopted the phrase. However, underlying all such perspectives is the foundational idea of representing knowledge using a graph abstraction, with nodes representing entities of interest in a given domain, and edges representing relations between those entities. Typically a knowledge graph will model different types of relations, which can be captured by labelled edges, with a different label used for each type of relation. Knowledge can consist of *simple assertions*, such as Charon orbits Pluto, which can be represented as directed labelled edges in a graph. Knowledge may also consist of *quantified assertions*, such as all stellar planets orbit stars, which require a more expressive formalism to capture, such as an ontology or rule language.

A reader familiar with related concepts may then question: what is the difference between a graph database and a knowledge graph, or an ontology and a knowledge graph? Per the previous definition, both graph databases[1] and ontologies can be considered knowledge graphs. It is then natural to ask: what, then, is new about knowledge graphs? What distinguishes research on knowledge graphs from research on graph databases, or ontologies, etc.?

If we so wish, we can choose to see the “vagueness” surrounding knowledge graphs as a feature not a bug: this vagueness offers flexibility for different communities to adapt the concept to their interests. No single research community can “lay claim” to the topic. Knowledge graphs can then become a commons for researchers from different areas to share and exchange techniques relating to the use of graphs to represent data and knowledge. In any case, most of the different choices – what graph model to use, what graph query language to use, whether to use rules or ontologies – end up being quite superficial choices.

Anecdotally, many of the most influential papers on knowledge graphs have emerged from the machine learning community, particularly relating to two main techniques: *knowledge graph embeddings* [68] and *graph neural networks* [71] (which we will discuss later). These works are not particularly related to graph databases (they do not consider regular expressions on paths, graph pattern matching, etc.) nor are they particularly related to ontologies (they do not consider interpretations, entailments, etc.), but by exploiting knowledge in the context of large-scale graph-structured data, they are related to knowledge graphs. Subsequently, one can find relations between these ostensibly orthogonal techniques; for example, while graph neural networks are used to inductively classify nodes in the graph based on numerical methods, ontologies can be used to deductively classify nodes in the graph based on symbolic methods, which raises natural questions about how they might compare.

Knowledge graphs, if we so choose, can be seen as an opportunity for researchers to bring together traditionally disparate techniques – relating to the use of graphs to represent and exploit knowledge – from different communities of Computer Science: to share and discuss them, to compare and contrast them, to unify and hybridise them. The goal of these notes will be to introduce examples of some of the principal directions along which such research may follow.

Recently we have published a tutorial paper online that offers a much more extensive (and generally less technical, more example-based) introduction to knowledge graphs than would be possible here, to which we refer readers new to the topic [35].[2] In the interest of keeping the current notes self-contained, and in order to establish notation, we will re-introduce some of the key concepts underlying knowledge graphs. However, our focus herein is to introduce specific open questions that we think may become of interest for knowledge graphs in the coming years. Specifically, we introduce six main concepts underlying knowledge graphs: data models, queries, ontologies, rules, embeddings and graph neural networks. Most of the questions we introduce intersect these topics.

[1] We intend to refer to the data model known as graph databases [9], not graph database systems [2].

[2] An abridged version of [35] is currently under review for ACM CSUR.

2 Data Model

Knowledge graphs assume a graph-structured data model. The high-level benefits of modelling data as graphs are as follows:

– Graphs offer a more intuitive abstraction of certain domains than alternative data models; for example, metro maps, flight routes, social networks, protein pathways, etc., are often visualised as graphs.
– When compared with (normalised) relational schemas, graphs offer a simpler and more flexible way to represent diverse, incomplete, evolving data, which can be defined independently of a particular schema.
– Graphs from different sources can be initially combined by taking the union of the constituent elements (nodes, edges, etc.).
– Unlike other semi-structured data models based on trees, graphs allow for representing and working with cyclic relations and do not assume a hierarchy.
– Graphs offer a more intuitive representation for querying and working with paths that represent indirect relations between entities.

A number of graph data models have become popular in practice, principally *directed edge-labelled graphs*, and *property graphs* [4]. The simpler of the two models is that of directed edge-labelled graphs, or simply *del graphs* henceforth. To build a del graph we will assume a set of constants $\mathbf{C}$.

Definition 1 (Directed edge-labelled graph). *A* directed edge-labelled graph *(or* del graph*) is a set of triples* $G \subseteq \mathbf{C} \times \mathbf{C} \times \mathbf{C}$.

We provide an example of a del graph in Figure 1, describing various astronomic bodies. The data are incomplete, but more data can be added incrementally by adding new nodes and edges. Each triple $(s, p, o) \in G$ denotes a directed labelled edge of the form $s \xrightarrow{p} o$. Nodes in the graph denote entities of interest, and edges between them denote (binary) relations. We see both directed and undirected cycles formed from the relationships present between the entities.

The property graph model is more ornate, and allows for adding labels and property–value pairs on both nodes and edges [4].

Definition 2 (Property graph). *A* property graph *G is a tuple*

$$G \coloneqq (V, E, L, P, U, e, l, p)$$

where $V \subseteq \mathbf{C}$ is a set of node ids, $E \subseteq \mathbf{C}$ is a set of edge ids, $L \subseteq \mathbf{C}$ is a set of labels, $P \subseteq \mathbf{C}$ is a set of properties, $U \subseteq \mathbf{C}$ is a set of values, $e : E \to V \times V$ maps an edge id to a pair of node ids, $l : V \cup E \to 2^L$ maps a node or edge id to a set of labels, and $p : V \cup E \to 2^{P \times U}$ maps a node or edge id to a set of property–value pairs.

In Figure 2 we provide an example of a property graph alongside a del graph representing analogous information. The graph is defined as follows:

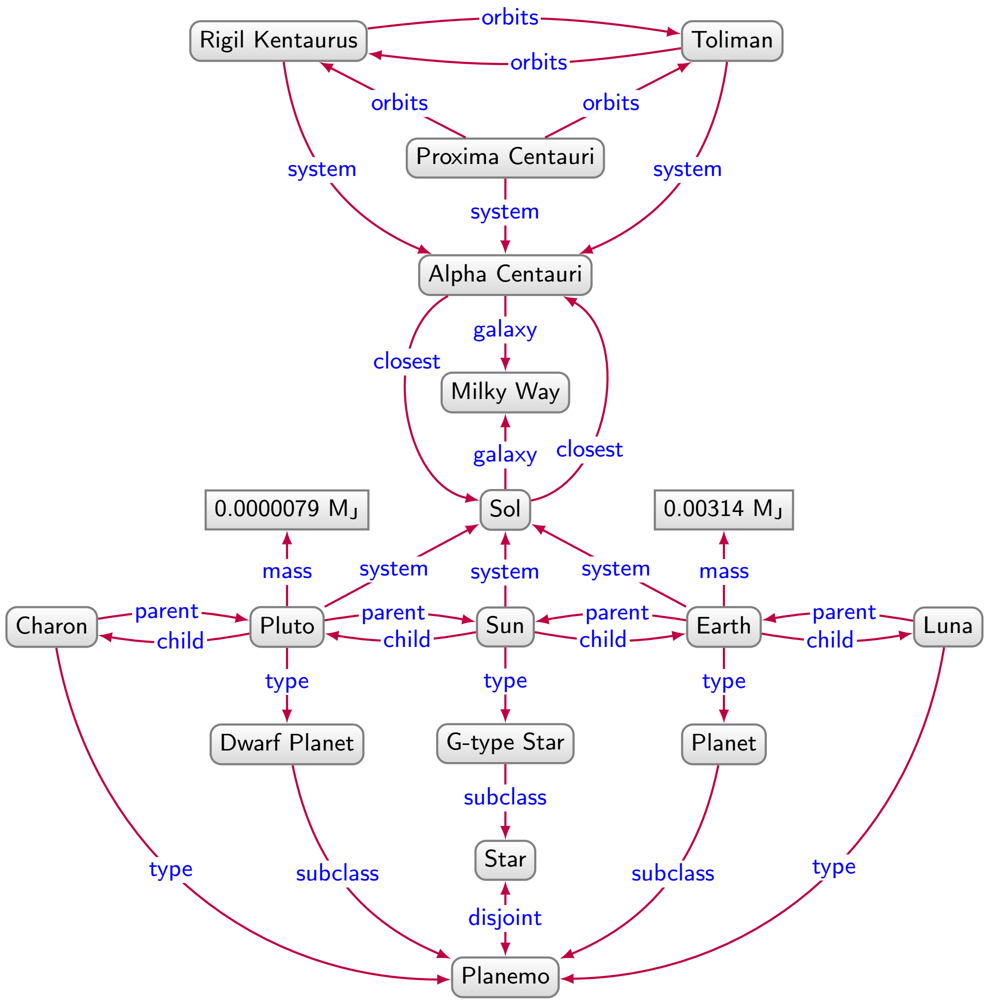


Fig. 1. Del graph describing astronomic bodies

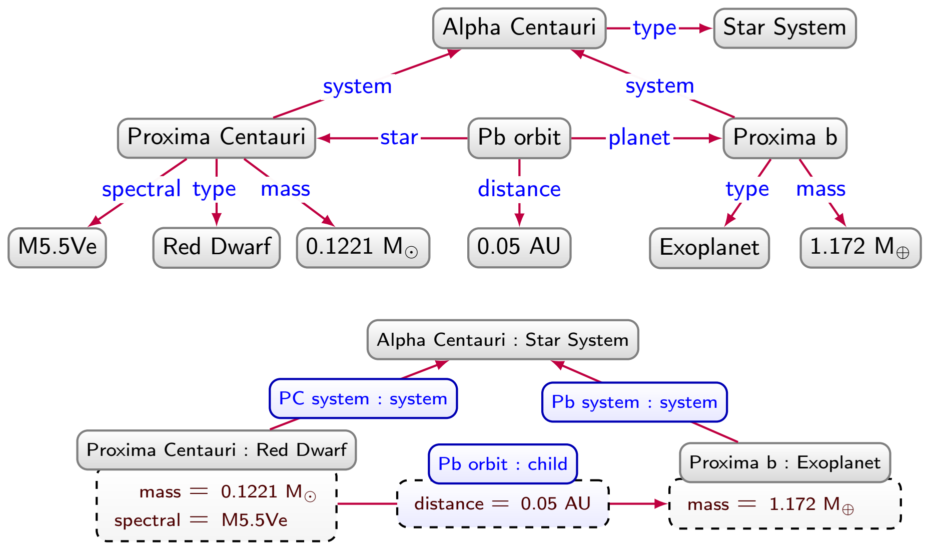


Fig. 2. Del graph (above) and property graph (below) with analogous information describing a planet orbiting Proxima Centauri

- $V \coloneqq \{$`Alpha Centauri`, `Proxima b`, `Proxima Centauri`$\}$
- $E \coloneqq \{$`Pb orbit`, `Pb system`, `PC system`$\}$
- $L \coloneqq \{$`Exoplanet`, `Red Dwarf`, `Star System`, `child`, `system`$\}$
- $P \coloneqq \{$`distance`, `mass`, `spectral`$\}$
- $U \coloneqq \{$`0.05 AU`, `0.1221 M`$_\odot$, `1.172 M`$_\oplus$, `M5.5Ve`, $\}$
- $e($`Pb orbit`$) \coloneqq ($`Proxima Centauri`, `Proxima b`$)$,
  $e($`Pb system`$) \coloneqq ($`Proxima b`, `Alpha Centauri`$)$,
  $e($`PC system`$) \coloneqq ($`Proxima Centauri`, `Alpha Centauri`$)$
- $l($`Alpha Centauri`$) \coloneqq \{$`Star System`$\}$, $l($`Proxima b`$) \coloneqq \{$`Exoplanet`$\}$,
  $l($`Proxima Centauri`$) \coloneqq \{$`Red Dwarf`$\}$, $l($`Pb orbit`$) \coloneqq \{$`child`$\}$,
  $l($`Pb system`$) \coloneqq \{$`system`$\}$, $l($`PC system`$) \coloneqq \{$`system`$\}$
- $p($`Alpha Centauri`$) \coloneqq \{\}$, $p($`Proxima b`$) \coloneqq \{($`mass`, `1.172 M`$_\oplus)\}$,
  $p($`Proxima Centauri`$) \coloneqq \{($`mass`, `0.1221 M`$_\odot), ($`spectral`, `M5.5Ve`$)\}$,
  $p($`Pb orbit`$) \coloneqq \{($`distance`, `0.05 AU`$)\}$, $p($`Pb system`$) \coloneqq \{\}$, $p($`PC system`$) \coloneqq \{\}$

While del graphs form the basis of the RDF data model [18], property graphs are used in a variety of popular graphs databases systems, such as Neo4j [48]. Comparing del graphs and property graphs, we can say that del graphs are simpler, but property graphs are more flexible, particularly in terms of the ability to add property–value pairs to edges. However, property graphs can be represented as del graphs (and vice versa) as illustrated in Figure 2, where edges with property–value pairs in the property graph can be converted to nodes in the del graph, as seen for `Pb orbit`; edges without property–value pairs in the property graph can be represented directly as edges in the del graph.

We identify two particular topics relating to modelling data as graphs.

Topic 2.1: Representing complex data

While graphs allow for representing many domains of data in an intuitive and concise manner, other forms of data are not so straightforward to represent. Being based on sets of elements, graphs are inherently unordered, which raises the question of how ordered data (lists, tuples, arrays, etc.) should be represented. Likewise statistical data are often associated with particular datatypes (integer, decimal, temporal, etc.), particular units (kilograms, pounds, seconds, hours, etc.), particular levels of precision (error bounds, rounding, etc.). An important topic then relates to modelling more complex forms of real-world data as graphs.

Thus far we have assumed graphs to be populated with a generic set of constants $\mathbf{C}$. However, knowledge graphs in practice may be populated with diverse terms; for example, `Proxima b` is an identifier for a planet, while `0.1221` $\mathtt{M}_\odot$ is a complex numerical quantity with an associated unit of measure (Solar masses). Recognising this, the RDF data model, based on del graphs, supports the following terms:

- *Internationalized Resource Identifiers* (IRIs) serve as identifiers for nodes and edge labels, which can be linked to from across the Web.
- *Literals* represent datatype values, such as strings (optionally with language tags), numeric values, booleans, dates, times, durations, etc.
- *Blank nodes* are not strictly speaking constants, but rather represent existential terms, used to represent entities known to exist but whose values are unknown, or for enabling syntactic shortcuts [34].

Still there are open questions regarding how units – such as $\mathtt{M}_\oplus$ (Earth masses), $\mathtt{M}_\mathtt{J}$ (Jupiter masses), $\mathtt{M}_\odot$ (Solar masses) – can be represented, or how levels of precision should be handled. Although proposals have been made to represent units and precision in graphs [16, 43], often knowledge graphs will adopt an ad hoc representation. There are also open questions regarding how to perform querying and reasoning over statistical data, with automatic translation of units and handling of precision.

Likewise for ordered data, although a number of proposals have been made for representing lists as a graph, these proposals are quite verbose [46].

A more general question arises: how should graph models be extended (if at all) in order to be able to concisely, intuitively and comprehensively model diverse, real-world data? Should we allow only simple terms and use complex graph structures to model such data, or should we support complex terms – e.g., allowing tuples, arrays, tables, trees, edges, graphs, etc., as nodes – to simplify the graph structure? How can we explore such trade-offs and evaluate different proposals? How can we apply querying, reasoning, machine learning, etc., on these different representations?

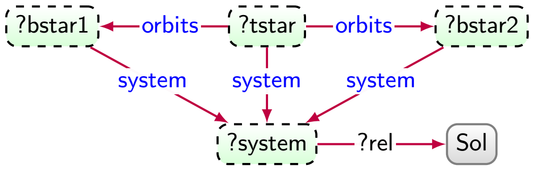


Fig. 3. Del graph pattern looking for trinary star systems related to Sol

Topic 2.2: Semantics of property graphs

While the semantics of del graphs have long been explored, the semantics of property graphs are less well understood. In reference to Figure 2, for example, we might ask if the child relation between `Proxima Centauri` and `Proxima b` holds. Intuitively we might assume that it does, but we should keep in mind that we may have property–value pairs on the edge that state `deprecated = true`, or `probability = 0.1`, for example. An open research question then relates to defining the semantics of property graphs, taking into consideration the semantics of labels and property–value pairs on nodes and edges. (We refer the interested reader to Attributed Description Logics [39], which make progress in this direction.)

Henceforth we will focus on del graphs, as the simpler of the two models.

3 Queries

We may wish to pose questions over the data represented by the graph. While a range of query languages have been proposed for querying graphs – including SPARQL for RDF graphs [29]; Cypher [23], Gremlin [57], and G-CORE [3] for property graphs – the same foundational elements underlie all such languages: *graph patterns*, *relational algebra*, and *path expressions* [4]. Herein we will assume data represented as a del graph, though the concepts we present generalise quite straightforwardly to other graph models, as appropriate [35].

At the core of graph queries is the notion of graph patterns. The most basic form of graph pattern is a graph that additionally allows *variables* to replace constants in any position. We refer to the set of variables as $\mathbf{V}$. We refer to the union of constants $\mathbf{C}$ and variables $\mathbf{V}$ as *terms*, denoted $\mathbf{T}$ (i.e., $\mathbf{T} = \mathbf{C} \cup \mathbf{V}$).

Definition 3 (Directed edge-labelled graph pattern). *A* directed edge-labelled graph pattern *(or* del graph pattern*) is a set of triples* $Q \subseteq \mathbf{T} \times \mathbf{T} \times \mathbf{T}$.

We provide an example in Figure 3 of a del graph pattern looking for trinary star systems related (in some way) to Sol. Variables are prefixed with "?". A graph pattern is then evaluated over a data graph by mapping the variables of the graph pattern to constants such that the image of the graph pattern under

Table 1. Evaluation of the del graph pattern of Figure 3 over the del graph of Figure 1 under homomorphism-based semantics

?bstar1	?bstar2	?tstar	?system	?rel
Rigil Kentaurus	Toliman	Proxima Centauri	Alpha Centauri	closest
Rigil Kentaurus	Rigil Kentaurus	Proxima Centauri	Alpha Centauri	closest
Toliman	Toliman	Proxima Centauri	Alpha Centauri	closest
Rigil Kentaurus	Rigil Kentaurus	Toliman	Alpha Centauri	closest
Toliman	Toliman	Rigil Kentaurus	Alpha Centauri	closest

the mapping is a sub-graph of the data graph. More formally, let $\mu : \mathbf{V} \to \mathbf{C}$ denote a partial mapping from variables to constants, and let $\mathrm{dom}(\mu)$ denote the set of variables for which μ is defined. Further let $\mathbf{V}(Q)$ denote the set of variables used in Q. We can then define the evaluation of Q over G as:

Definition 4 (Evaluation of a del graph pattern). *Let Q be a del graph pattern and let G be a del graph. We then define the* evaluation of graph pattern Q over G, *as* $Q(G) \coloneqq \{\mu \mid \mu(Q) \subseteq G$ *and* $\mathrm{dom}(\mu) = \mathbf{V}(Q)\}$.

Graph patterns of this form can be seen as transforming graphs into tables. In Table 1 we present the results of evaluating the del graph pattern of Figure 3 over the del graph of Figure 1, where each row indicates a *solution* given by a mapping μ per the previous definition. Graph patterns can be evaluated under different semantics by placing further restrictions on μ. Without further restrictions, a *homomorphism-based semantics* is considered where two or more variables in an individual solution can be mapped to a single constant in the data. On the other hand, if we restrict μ to be one-to-one, a (sub-graph) *isomorphism-based semantics* is considered, where each variable in an individual solution must map to a different constant. All of the solutions shown in Table 1 are returned under the unrestricted homomorphism-based semantics, while only the first solution is returned under the restricted isomorphism-based semantics.

Given that graph patterns return tables, a natural idea is to introduce the relational algebra to allow the solutions of a graph pattern to be transformed, and the solutions of several graph patterns to be combined: using π to project variables or σ to select rows returned from a single graph pattern, or using $\cup$, $-$, or $\bowtie$ to apply a union, difference, or join (respectively) across the results of two graph patterns. We can also introduce further syntactic operators, such as left-joins ($⟕$, aka optionals), anti-joins ($\rhd$; aka not exists), etc. As a simple example, given the graph pattern Q of Figure 3, we may define $\pi_{\mathsf{?tstar}}(\sigma_{\mathsf{?bstar1} \neq \mathsf{?bstar2}}(Q))$ to select only solutions where ?bstar1 and ?bstar2 map to different constants, and subsequently project only the results for the ?tstar variable. Graph patterns extended with the relational algebra are called *complex graph patterns* [3]. Graph patterns extended with projection alone are called *conjunctive queries*.

The features we have seen thus far can only match bounded sub-graphs of the data graph (more formally, we can say that they are Gaifman-local [45]). However, we may be interested to find pairs of nodes that are connected by paths of arbitrary lengths. In order to express such queries, we can introduce *path expressions* and *regular path queries*.[3]

Definition 5 (Path expression). *A constant $c \in \mathbf{C}$ is a* path expression*. Furthermore:*

- *If r is a path expression, then r^- (inverse) and r^* (Kleene star) are* path expressions.
- *If r_1 and r_2 are path expressions, then $r_1 \cdot r_2$ (concatenation) and $r_1 \mid r_2$ (disjunction) are* path expressions.

In the following we assume that the evaluation of a path expression returns pairs of nodes connected by some path that satisfies the path expression [29]. Other query languages may support returning paths [23, 3]. We will further introduce the notation N_G to denote the nodes of G; more formally, we define the nodes of G as $N_G \coloneqq \{x \mid \exists p, y : (x,p,y) \in G \text{ or } (y,p,x) \in G\}$.

Definition 6 (Path expression evaluation). *Given a del graph G and a path expression r, we define the* evaluation of r over G*, denoted $r[G]$, as follows:*

$$\begin{aligned} r[G] &\coloneqq \{(u,v) \mid (u,r,v) \in E\} \text{ (where } r \in \mathbf{C}) \\ r^-[G] &\coloneqq \{(u,v) \mid (v,u) \in r[G]\} \\ r_1 \mid r_2[G] &\coloneqq r_1[G] \cup r_2[G] \\ r_1 \cdot r_2[G] &\coloneqq \{(u,v) \mid \exists w \in N_G : (u,w) \in r_1[G] \text{ and } (w,v) \in r_2[G]\} \\ r^*[G] &\coloneqq N_G \cup \bigcup_{n \in \mathbb{N}^+} r^n[G] \end{aligned}$$

where by r^n we denote the n^{th}-concatenation of r (e.g., $r^3 = r \cdot r \cdot r$).

Let $\mathbf{R}$ denote the set of all path expressions. We next define a *regular path query* and a *navigational graph pattern* [4].

Definition 7 (Regular path query). *A* regular path query *is a triple $(x,r,y) \in \mathbf{T} \times (\mathbf{R} \cup \mathbf{V}) \times \mathbf{T}$.*

Definition 8 (Navigational graph pattern). *A* navigational graph pattern *Q is a set of regular path queries.*

We provide an example of a navigational graph pattern in Figure 4, searching for planemos in the Milky Way, along with the star(s) they orbit. The evaluation of a navigational graph pattern is then defined analogously to the evaluation of graph patterns, but where pairs of nodes connected by arbitrary-length paths – rather than simply edges – can now be matched. The navigational graph pattern of Figure 4 will return four results, with ?star mapped to Sun in each, and ?planemo mapped to Earth, Pluto, Charon and Luna.

[3] We implicitly refer to *two-way* regular path queries as inverse expressions are quite widely used in practice [15].

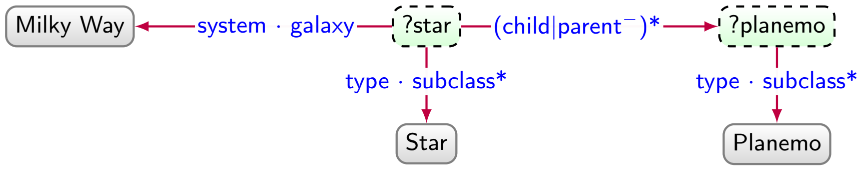


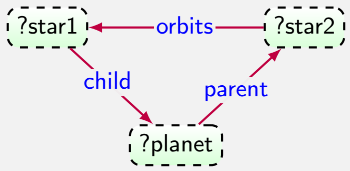
Fig. 4. Navigational graph pattern looking for planemos in the Milky Way and the stars that they orbit

Finally, noting that navigational graph patterns again transform graphs into tables, it is natural to define *complex navigational graph patterns*, which support applying relational algebra over the results of one or more navigational graph patterns. Navigational graph patterns extended with (only) projection are known as *conjunctive two-way regular path queries* (C2RPQs) [9].

The task of finding solutions for a query over a graph is known as *query answering*. While query answering have been well-studied in the literature [9, 4], there are a number of interesting problems that arise when considering the evaluation of (complex) navigational graph patterns. We now discuss two topics.

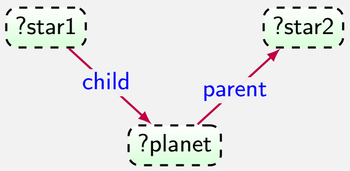
Topic 3.1: *-case optimality

In the context of relational databases, the Atserias–Grohe–Marx (AGM) bound establishes a tight upper bound for the number of tuples that can be returned for conjunctive queries (without projection) over relations with a given number of tuples [7]. This bound can be naturally extended to give a tight bound in the case of graph patterns. For example, consider the following triangular graph pattern looking for planets of binary stars:



The AGM bound is based on a *fractional edge cover*, where each edge (relation) is assigned a weight in $[0,1]$ such that, for each variable, when we sum the weights of the edges it appears in, the value is at least 1. In this case, for example, we can assign $\frac{1}{2}$ to orbits, $\frac{1}{2}$ to child and $\frac{1}{2}$ to parent, which is sufficient to (fractionally) cover all variables. The number of results returned by the query is then bounded by $n_{\mathsf{o}}^{w_{\mathsf{o}}} \cdot n_{\mathsf{c}}^{w_{\mathsf{c}}} \cdot n_{\mathsf{p}}^{w_{\mathsf{p}}}$, where n_{o}, n_{c}, n_{p} denote the size of the relations for orbits, child and parent, respectively, while w_{o}, w_{c}, w_{p} denote their fractional edge cover weights. If we find the fractional edge cover that minimises the aforementioned product over all relations, then the bound is tight. In this case, the tight bound is given by $n_{\mathsf{o}}^{\frac{1}{2}} \cdot n_{\mathsf{c}}^{\frac{1}{2}} \cdot n_{\mathsf{p}}^{\frac{1}{2}}$; assuming $n = n_{\mathsf{o}} = n_{\mathsf{c}} = n_{\mathsf{p}}$ for simplicity, we get $n^{\frac{3}{2}}$.

The AGM bound then gives us a general mechanism by which we can estimate the maximum number of results that a graph pattern can return. When we think of methods for enumerating all of the solutions for a graph pattern Q over a given graph, the best we can then hope for is an algorithm that runs in $O(\beta(Q))$, where $\beta(Q)$ is the AGM bound for Q. This is not trivial as the classical strategies for evaluating the aforementioned graph pattern (nested loops, hash joins, etc.) would be to proceed pattern-by-pattern, which would first execute a join such as:



But in this case we must assign 1 to child and 1 to parent to have a fractional edge cover, meaning that the bound for this join would be n^2, which could not be processed within the $O(n^{\frac{3}{2}})$ runtime.

However, there do exist *worst-case optimal join algorithms* [50] that allow for the solutions of such queries to always be evaluated within linear time of the worst-case output size (the AGM bound). The core idea of such algorithms is to apply multiple joins at once, essentially evaluating variable-by-variable, rather than pattern-by-pattern. Such algorithms have been implemented in the context of querying graphs, and have shown that these theoretic guarantees translate into promising results in practice, particularly for more complex graph patterns with cycles (e.g., [51, 1, 38, 36]).

More work is left to do in terms of extending worst-case optimal algorithms to offer similar guarantees when other features of graph queries are present, including path expressions. Furthermore, works that go beyond worst-case optimality, perhaps considering average-case output size, or even instance-optimal join algorithms, are left to explore [50].

Topic 3.2: Native graph querying

Graph patterns return tables of solutions rather than graphs. An interesting research topic to explore is then to consider query languages based on composable operators that transform and return native graph objects – i.e., nodes, edges, paths, graphs, etc. – rather than tables.

We defined the evaluation of regular path queries such that the nodes connecting paths satisfying the path expression are returned. In practice, it may often be necessary to return not only pairs of nodes, but also the paths themselves that are matched by the expression. Unlike pairs of nodes, paths in the presence of cycles may be infinite! For this reason, often a particular restriction is applied to the paths considered, such as to include only *simple paths*: paths that only visit each node at most once. As an additional complication, differing semantics may affect the complexity of evaluation in non-trivial ways; for example, finding a simple path of even length between two nodes (e.g., matching (orbits·orbits)* in a directed graph is NP-complete [42]). Similarly, the number of even simple paths can be astronomically large, even for small graphs, making them difficult to implement [5]. Finally, it is unclear how paths of arbitrary length should be represented in the results, or what kinds of operators over paths can be defined in the query language; for example, how should we join different tables on a variable mapped to paths?

Practical query languages like Cypher [23] support returning paths, and offer some operators, such as `length(·)` over paths. G-CORE [3] takes this one step further, and treats paths as "first-class citizens", allowing for returning paths in results, finding weighted shortest paths, and more besides. Still, however, more work is needed to understand how paths in results can be represented, what key operators should be defined for them, what are the appropriate semantics to apply, etc.

On the other hand, we may wish to return graphs from queries. Along these lines, SPARQL and G-CORE support `CONSTRUCT` queries, which allow for transforming a table of intermediate solutions into an output graph [29, 3]. Recursive extensions of `CONSTRUCT` queries have further been proposed and studied in the literature [56]. Still, however, `CONSTRUCT` queries are based on graph patterns that generate tables of intermediate results.

We then pose the idea of a *native graph query language* that is composed of unary and binary operators over graphs that return graphs. In the case of del graphs, which are defined as sets of triples, *selection* can be trivially defined in terms of filtering such triples, while set-based operators – *union*, *intersection*, *difference* – can be defined naturally with respect to the sets of triples. An interesting question relates to how joins on two graphs could or should be defined. Are joins even needed? What about aggregation? Would a graph-based definition of operators affect the computational complexity of query answering? Would it have practical benefits?

4 Ontologies

Looking more closely at the graph of Figure 1, we may be able to deduce additional knowledge beyond what is explicitly stated in the graph. For example, we might conclude that Charon and Luna are instances of Moon; that Toliman and Rigil Kentaurus are instances of Binary Stars; that Luna, Earth, Sun, etc., are all part of the Milky Way, etc. In order to be able to draw such conclusions automatically from a graph, we require a formal representation of the semantics of the terms used in the graph. While there are a number of alternatives that we might consider for this task, we initially focus on *ontologies*, which have long been studied in the context of defining semantics over graphs.

Ontologies centre around three main types of elements: individuals, concepts (aka classes) and roles (aka properties). They allow for making formal, well-defined claims about such elements, which in turn opens up the possibility to perform automated reasoning. In terms of the semantics of ontologies, there are many well-defined syntaxes we can potentially use, but perhaps the most established formalism is taken from *Description Logics* (*DL*) [58, 41, 8], which studies logics centring around (unary and) binary relations, and logical primitives for which reasoning tasks remain decidable.

Table 2 defines the typical DL constructs one can find in the literature. The syntax column denotes how the construct is expressed in DL. A DL ontology then consists of an assertional box (A-Box), a T-Box (terminological box), and an R-Box (role box), each of which consists of a set of axioms that describe formal claims about individuals, concepts and properties, respectively.

Definition 9 (DL ontology). *A* DL ontology O *is defined as a tuple* $(\mathsf{A}, \mathsf{T}, \mathsf{R})$*, where* A *is the* A-Box*: a set of assertional axioms;* T *is the* T-Box*: a set of concept (aka class/terminological) axioms; and* R *is the* R-Box*: a set of role (aka property/relation) axioms.*

Syntactically, DL ontologies can be serialised as (del) graphs. Such a serialisation is defined by the Web Ontology Language (OWL) [31], which draws inspiration from the DL area towards standardising languages for which common reasoning tasks are decidable (or even tractable), and which defines serialisations of OWL ontologies as RDF graphs. In this context, looking at Figure 1, for example, a triple such as (Planet, subclass, Planemo) can be read as a concept axiom $\texttt{Planet} \sqsubseteq \texttt{Planemo}$, a triple such as (Pluto, child, Charon) can be read as a role axiom $\texttt{child}(\texttt{Pluto}, \texttt{Charon})$, while a triple such as (Charon, type, Planemo) can be read as a concept assertion $\texttt{Planemo}(\texttt{Charon})$.

Regarding the formal meaning of these axioms, the semantics column of Table 2 defines axioms using *interpretations*.

Definition 10 (DL interpretation). *A* DL interpretation I *is defined as a pair* $(\Delta^I, \cdot^I)$*, where* Δ^I *is the interpretation domain, and* $\cdot^I$ *is the interpretation function. The interpretation domain is a set of individuals. The interpretation function accepts a definition of either an individual* a*, a concept* C*, or a role* R*, mapping them, respectively, to an element of the domain* ($a^I \in \Delta^I$)*, a subset of the domain* ($C^I \subseteq \Delta^I$)*, or a set of pairs from the domain* ($R^I \subseteq \Delta^I \times \Delta^I$)*.*

An interpretation I *satisfies* an ontology O if and only if, for all axioms in O, the corresponding semantic conditions in Table 2 hold for I. In this case, we call I a *model* of O. This notion of a model gives rise to the key notion of entailment.

Definition 11. *Given two DL ontologies* O_1 *and* O_2*, we define that* O_1 entails O_2*, denoted* $\mathsf{O}_1 \models \mathsf{O}_2$*, if and only if every model of* O_1 *is a model of* O_2*.*

Table 2. Description Logic syntax and semantics

Name	**Syntax**	**Semantics** ($\cdot^I$)
	CONCEPT DEFINITIONS	
Atomic Concept	A	A^I (a subset of Δ^I)
Top Concept	$\top$	Δ^I
Bottom Concept	$\bot$	$\emptyset$
Concept Negation	$\neg C$	$\Delta^I \setminus C^I$
Concept Intersection	$C \sqcap D$	$C^I \cap D^I$
Concept Union	$C \sqcup D$	$C^I \cup D^I$
Nominals	$\{a\}$	$\{a^I\}$
Existential Restriction	$\exists R.C$	$\{x \mid \exists y : (x,y) \in R^I \text{ and } y \in C^I\}$
Universal Restriction	$\forall R.C$	$\{x \mid \forall y : (x,y) \in R^I \text{ implies } y \in C^I\}$
Self Restriction	$\exists R.\mathsf{Self}$	$\{x \mid (x,x) \in R^I\}$
Number Restriction	$\star\, n\, R$ (where $\star \in \{\geq, \leq, =\}$)	$\{x \mid \#\{y : (x,y) \in R^I\} \star n\}$
Qualified Number Restriction	$\star\, n\, R.C$ (where $\star \in \{\geq, \leq, =\}$)	$\{x \mid \#\{y : (x,y) \in R^I \text{ and } y \in C^I\} \star n\}$
	CONCEPT AXIOMS (T-Box)	
Concept Inclusion	$C \sqsubseteq D$	$C^I \subseteq D^I$
	ROLE DEFINITIONS	
Role	R	R^I (a subset of $\Delta^I \times \Delta^I$)
Inverse Role	R^-	$\{(y,x) \mid (x,y) \in R^I\}$
Universal Role	U	$\Delta^I \times \Delta^I$
	ROLE AXIOMS (R-Box)	
Role Inclusion	$R \sqsubseteq S$	$R^I \subseteq S^I$
Complex Role Inclusion	$R_1 \circ ... \circ R_n \sqsubseteq S$	$R_1^I \circ ... \circ R_n^I \subseteq S^I$
Transitive Roles	$\mathsf{Trans}(R)$	$R^I \circ R^I \subseteq R^I$
Functional Roles	$\mathsf{Func}(R)$	$\{(x,y),(x,z)\} \subseteq R^I$implies $y = z$
Reflexive Roles	$\mathsf{Ref}(R)$	for all $x \in \Delta^I : (x,x) \in R^I$
Irreflexive Roles	$\mathsf{Irref}(R)$	for all $x \in \Delta^I : (x,x) \notin R^I$
Symmetric Roles	$\mathsf{Sym}(R)$	$R^I = (R^-)^I$
Asymmetric Roles	$\mathsf{Asym}(R)$	$R^I \cap (R^-)^I = \emptyset$
Disjoint Roles	$\mathsf{Disj}(R,S)$	$R^I \cap S^I = \emptyset$
	ASSERTIONAL DEFINITIONS	
Individual	a	a^I (an element of Δ^I)
	ASSERTIONAL AXIOMS (A-Box)	
Role Assertion	$R(a,b)$	$(a^I, b^I) \in R^I$
Negative Role Assertion	$\neg R(a,b)$	$(a^I, b^I) \notin R^I$
Concept Assertion	$C(a)$	$a^I \in C^I$
Equality	$a = b$	$a^I = b^I$
Inequality	$a \neq b$	$a^I \neq b^I$

As an example of a DL ontology, for $\mathsf{O} := (\mathsf{A}, \mathsf{T}, \mathsf{R})$, let:

- $\mathsf{A} := \{\mathtt{Planet(Earth)}, \mathtt{parent(Luna,Earth)}, \mathtt{parent(Pluto,Sun)}\}$;
- $\mathsf{T} := \{\mathtt{Planet} \sqsubseteq \forall\mathtt{child.Moon}, \mathtt{Planemo} \equiv \mathtt{Moon} \sqcup \mathtt{Planet} \sqcup \mathtt{DwarfPlanet}\}$;
- $\mathsf{R} := \{\mathtt{parent} \sqsubseteq \mathtt{orbits}, \mathtt{parent} \equiv \mathtt{child}^{-}\}$.

For $I = (\Delta^I, \cdot^I)$, let:

- $\Delta^I := \{\oplus, \leftmoon, \pluto, \sun\}$;
- $\mathtt{Earth}^I := \oplus$, $\mathtt{Luna}^I := \leftmoon$, $\mathtt{Pluto}^I := \pluto$, $\mathtt{Sun}^I := \sun$;
- $\mathtt{Planet}^I := \{\oplus, \pluto\}$, $\mathtt{Planemo}^I := \{\oplus, \pluto, \leftmoon\}$;
- $\mathtt{parent}^I := \{(\leftmoon, \oplus), (\pluto, \sun)\}$, $\mathtt{child}^I := \{(\oplus, \leftmoon), (\sun, \pluto)\}$.

The interpretation I is not a model of O since it does not have that $\leftmoon$ is an instance of $\mathtt{Moon}$, nor that $\leftmoon$ orbits $\oplus$, nor that $\pluto$ orbits $\sun$. However, if we extend the interpretation I with the following:

- $\mathtt{Moon}^I := \{\leftmoon\}$;
- $\mathtt{orbits}^I := \{(\leftmoon, \oplus), (\pluto, \sun)\}$.

then the interpretation of I will satisfy – i.e., will be a model of – the ontology O. Notably even though the ontology O does not entail that $\mathtt{Planemo(Pluto)}$, it is still the case that I satisfies O; this is often referred to as the *Open World Assumption*, where ontologies are not assumed to completely describe the world, but rather to be incomplete. Now given the ontology:

$$\mathsf{O}' := (\{\mathtt{Moon(Luna)}\}, \{\mathtt{Moon} \sqsubseteq \mathtt{Planemo}\}, \{\}) ,$$

we may – with a bit of thought – convince ourselves of the fact that any model I of O must also be a model of O', and hence that $\mathsf{O} \models \mathsf{O}'$.

Unfortunately, deciding entailment for DL ontologies using all of the axioms of Table 2 in an unrestricted manner is undecidable. Hence, different DLs then apply different restrictions to the use of these axioms in order to achieve particular guarantees with respect to the complexity of deciding entailment. Most DLs are founded on one of the following base DLs (we use indentation to denote that the child DL extends the parent DL):

$\mathcal{ALC}$ (*$\mathcal{A}$ttributive $\mathcal{L}$anguage with $\mathcal{C}$omplement* [61]), supports atomic concepts, the top and bottom concepts, concept intersection, concept union, concept negation, universal restrictions and existential restrictions. Role and concept assertions are also supported.

  $\mathcal{S}$ extends $\mathcal{ALC}$ with transitive closure.

These base languages can be extended as follows:

$\mathcal{H}$ adds role inclusion.

  $\mathcal{R}$ adds (limited) complex role inclusion, as well as role reflexivity, role irreflexivity, role disjointness and the universal role.

$\mathcal{O}$ adds nominals.
$\mathcal{I}$ adds inverse roles.
$\mathcal{F}$ adds (limited) functional properties.
  $\mathcal{N}$ adds (limited) number restrictions.
    $\mathcal{Q}$ adds (limited) qualified number restrictions (subsuming $\mathcal{N}$ given $\top$).

We use "(limited)" to indicate that such features are often only allowed under certain restrictions to ensure decidability; for example, complex roles (chains) typically cannot be combined with cardinality restrictions. DLs are then typically named per the following scheme, where $[a|b]$ denotes an alternative between a and b and $[c][d]$ denotes a concatenation cd:

$$[\mathcal{ALC}|\mathcal{S}][\mathcal{H}|\mathcal{R}][\mathcal{O}][\mathcal{I}][\mathcal{F}|\mathcal{N}|\mathcal{Q}]$$

Examples include $\mathcal{ALCO}$, $\mathcal{ALCHI}$, $\mathcal{SHIF}$, $\mathcal{SROIQ}$, etc. These languages often apply additional restrictions on concept and role axioms to ensure decidability, which we do not discuss here. For further details on Description Logics, we refer to the recent book by Baader et al. [8].

We now discuss a research topic relating to the combination of ontology entailment and graph querying.

Topic 4.1: OBDA on graphs

The area of Ontology-Based Data Access (OBDA) [72] focuses on finding and implementing DL fragments for which query answering can be conducted over an ontology by means of a *query rewriting strategy*. Given an ontology and a graph query, this strategy involves extending the query such that, when it is evaluated over the base data, the solutions include those also given by entailments with respect to the ontology. For example, with respect to the aforementioned example ontology O, a simple graph pattern $\{(\mathsf{?moon}, \mathsf{type}, \mathsf{Moon})\}$ may be rewritten to query for:

$$\begin{aligned} &\{(\mathsf{?moon}, \mathsf{type}, \mathsf{Moon})\}\cup \\ \{(\mathsf{?planet}, \mathsf{type}, \mathsf{Planet})&(\mathsf{?planet}, \mathsf{child}, \mathsf{?moon})\}\cup \\ \{(\mathsf{?planet}, \mathsf{type}, \mathsf{Planet})&(\mathsf{?moon}, \mathsf{parent}, \mathsf{?planet})\} \end{aligned}$$

capturing the various ways in which instances of Moon could be entailed from the base data. The ability of an ontology language (e.g., DL fragment) to have all of its entailments supported in this way for queries using the basic relational algebra (i.e., complex graph patterns) is called *first-order rewritability* (referring to "first-order queries" which are equivalent to relational algebra queries per Codd's theorem). In general, only heavily restricted ontology languages have this property [6], often only conjunctive queries are accepted, and often the rewritten queries are unions of graph patterns (rather than using the full algebra) as illustrated above.

So what happens if the input query is not simply a graph pattern or a conjunctive query, but is rather a (complex) navigational graph pattern? A number of works have studied the complexity of query answering for navigational graph patterns in the presence of ontologies of varying expressivity [54, 65, 12]. However, such queries are not first-order queries, and are not first-order rewritable, where it is thus unclear how regular path queries can be supported in the context of OBDA, for example.

Many database systems, however, support features that go beyond first-order queries; for example, relational database engines support recursion, while graph database engines support path expressions. Such features of query engines have rarely exploited as targets for query rewriting strategies; exceptions include works on supporting transitive closure [63] and recursive rules [73] using SQL recursion, and studies of other forms of rewritability, such as rewriting to Datalog and Monadic Disjunctive Datalog [22].

This leaves the question: given a graph database system capable of answering (complex) navigational graph patterns – as popular in practice – what are the limits to the types of OBDA that the system can support through query rewriting? What kinds of input queries can be supported with respect to which ontology languages? A number of works have explored query rewriting strategies considering navigational queries as targets, but have focussed on particular ontology languages [13, 21]. An interesting topic would be to explore, more generally, what kinds of ODBA typical graph database engines can support, and under what assumptions. Furthermore, experimental works to understand the limitations of such techniques in terms of scalability and performance would be of importance to understand how such systems can be adopted in practice.

5 Rules

Another way to define the semantics of knowledge graphs is to use *rules*. Rules can be formalised in many ways – as Horn clauses, as Datalog, etc. – but in essence they consist of IF-THEN statements, where IF some condition holds, THEN some entailment holds. Here we define rules based on graph patterns.

Definition 12 (Rule). *A* rule *is a pair $R := (B, H)$ such that B and H are graph patterns. The graph pattern B is called the body of the rule while H is called the head of the rule.*

Given a graph G and a rule $R = (B, H)$, we can then apply R over G by taking the union of $\mu(H)$ for all $\mu \in B(G)$. Typically we will assume that the set of variables used in H will be a subset of those used in B ($\mathbf{V}(H) \subseteq \mathbf{V}(B)$), which assures us that $\mu(H)$ will always result in a graph without variables. Given a set of rules, we can then apply each rule recursively, accumulating the results in the graph until a fixpoint is reached, thus enriching our graph with entailments.

There is a large intersection between the types of semantics we can define with rules and with DL-based ontologies [40]. For example, we can capture the role-inclusion axiom $\mathsf{parent} \sqsubseteq \mathsf{orbits}$ as a rule (written $B \rightarrow H$):

$$\{(\texttt{?x}, \texttt{parent}, \texttt{?y})\} \rightarrow \{(\texttt{?x}, \texttt{orbits}, \texttt{?y})\}$$

or we can capture the concept inclusion $\texttt{Planet} \sqsubseteq \forall\texttt{child}.\texttt{Moon}$ as:

$$\{(\texttt{?x}, \texttt{type}, \texttt{Planet}), (\texttt{?x}, \texttt{child}, \texttt{?y})\} \rightarrow \{(\texttt{?y}, \texttt{type}, \texttt{Moon})\} \ .$$

However, we cannot capture axioms such as $\texttt{BinaryStar} \sqsubseteq \exists\texttt{orbits}.\texttt{BinaryStar}$ with the types of rules we have seen. Such an axiom would need a rule like:

$$\{(\texttt{?x}, \texttt{type}, \texttt{BinaryStar})\} \rightarrow \exists y : \{(\texttt{?x}, \texttt{orbits}, y), (y, \texttt{type}, \texttt{BinaryStar})\}$$

where the head introduces a fresh existential variable y for each binary star, which itself is a binary star. (Note that if left unrestricted, such a rule, applied recursively on a single instance of a binary star, might end up creating an infinite chain of existential binary stars, each orbiting their existential successor!)

Additionally, rules as we have previously defined cannot capture axioms of the form $\texttt{Planemo} \sqsubseteq \texttt{Moon} \sqcup \texttt{Planet} \sqcup \texttt{DwarfPlanet}$ either. For this we would need a rule along the lines of the following:

$$\begin{aligned}\{(\texttt{?x}, \texttt{type}, \texttt{Planemo})\} \rightarrow & \{(\texttt{?x}, \texttt{type}, \texttt{Moon})\} \vee \\ & \{(\texttt{?x}, \texttt{type}, \texttt{Planet})\} \vee \\ & \{(\texttt{?x}, \texttt{type}, \texttt{DwarfPlanet})\} \ .\end{aligned}$$

Here we introduce the disjunctive connective "$\vee$" to state that when the body is matched, then at least one of the heads holds true.

Rules extended with existentials and disjunction have, however, been explored as extensions of Datalog. Datalog$^{\pm}$ variants support existential rules (aka *tuple generating dependencies* (*tgds*)) that allow for generating existentials in the head of the rule, per the `BinaryStar` example, but typically in a restricted way to ensure termination. On the other hand, Disjunctive Datalog allows for using disjunction in the head of rules, as seen in the `Planemo` example. These extensions allow rules to capture semantics similar to more expressive DLs [59, 26, 27].

On the other hand, rules can express entailments not typically supported in DLs, where a simple example of such a rule is:

$$\{(\texttt{?x}, \texttt{orbits}, \texttt{?y}), (\texttt{?y}, \texttt{orbits}, \texttt{?x})\} \rightarrow \{(\texttt{?x}, \texttt{sibling}, \texttt{?y})\} \ .$$

This type of entailment would require role intersection, which is not typically supported by DLs. More generally, cyclical graph patterns that entail role assertions are typically not supported in DLs, though easily captured by rules.[4]

Efficient reasoning systems have then been developed to support such rules over knowledge graphs, including most recently, Vadalog [11] and VLog [17].

[4] Cyclical graph patterns that entail *concept* assertions can be captured, in a slightly roundabout way, in DLs with Self Restrictions and Complex Role Inclusions.

Topic 5.1: Existential/disjunctive rule mining

Given a diverse knowledge graph, such as Wikidata [67], manually defining all of the rules that may hold is a costly exercise. To help arrive at an initial set of rules in such cases, a number of *rule mining* techniques have been proposed that, given a graph, a certain *support* threshold, and a certain *confidence* threshold, will return rules that meet the defined thresholds; support is defined as the number of entailments given by the rule that are deemed correct, while confidence is the ratio of entailments given by the rule that are deemed correct. Given that knowledge graphs are incomplete, while we may assume that the triples given by the graph are correct, it is not clear how we might know which triples are incorrect. A common heuristic is to apply a *Partial Completeness Assumption* (*PCA*) [24], where a triple (s, p, o) is considered correct if it appears in G, and incorrect if it does not appear in G but there exists a triple (s, p, o') that does appear in G; other triples are ignored. A seminal rule mining system along these lines is AMIE [24], which incrementally builds rules using a sequence of steps called "refinements", filtering rules that do not meet the specified thresholds. A number of later systems support additional features in rules, such as forms of negation [32]. However, to the best of our knowledge, mining existential or disjunctive rules remains open for knowledge graphs.

6 Context

All of the data represented in Figure 1 holds true with respect to a given *context*: a given scope of truth. As an example of temporal context, the Sun will cease being a G-type Star and will eventually become a White Dwarf in its later years. We may also have spatial context (e.g., to state that something holds true in a particular region of space, such as a solar eclipse affecting parts of Earth), provenance (e.g., to state that something holds true with respect to a given source, such as the mass of a given planet), and so forth.

There are a variety of syntactic ways to embed contextual meta-data in a knowledge graph. Within del graphs, for example, the options include reification [18], n-ary relations [18], and singleton properties [52]. Context can also be represented syntactically using higher-arity models, such as property graphs [4], RDF* [30], or named graphs [18]. For further details on these options for representing context in graphs, we refer to the extended tutorial [35].

Aside from syntactic conventions for representing context in graphs, an important issue is with respect to how the semantics of context should be defined. A general contextual framework for graphs based on *annotations* has been proposed by Zimmermann et al. [76], based on the notion of an *annotation domain*.

Definition 13 (Annotation domain). *Let A be a set of* annotation values. *An* annotation domain *is defined as an idempotent, commutative semi-ring $D = \langle A, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$.*

For example, we may define A to be powerset of numbers from $\{1, \ldots, 365\}$ indicating different days of the year. We may then annotate triples such as $(\mathsf{Luna}, \mathsf{occults}, \mathsf{Sun})$ with a set of days in a given year – say $\{13, 245, 301\}$ on which the given occultation will occur. Given another similar such triple – say $(\mathsf{Luna}, \mathsf{occults}, \mathsf{Jupiter})$ annotated with $\{46, 245, 323\}$ – we can define $\oplus$ to be $\cup$ and use it to find the annotation for days when Luna occults the Sun or Jupiter ($\{13, 46, 245, 301, 323\}$); we can further define $\otimes$ to be $\cap$ and use it to find the annotation for days when Luna occults the Sun and Jupiter ($\{245\}$). In this scenario, $\mathbf{0}$ would then refer to the empty set, while $\mathbf{1}$ would refer to the set of all days of the year. Custom annotation domains can be defined for custom forms of context, and can be used in the context of querying and rules for computing the annotations corresponding to particular results and entailments.

In the context of ontologies, a more declarative framework for handling context – called *contextual knowledge repositories* – was proposed by Homola and Serafini [37] based on Description Logics, with a related framework more recently proposed by Schuetz et al. [62] based on an OLAP-style abstraction. These frameworks can assign graphs into different hierarchical contexts, which can then be aggregated into higher-level contexts.

Still, there are a number of questions that remain open regarding context.

Topic 6.1: Complex contexts

Context – in the sense of the scope of truth – is an almost arbitrarily complex subject. To take some examples, context can be recurrent, relative, and conceptual. Such forms of context are not – to the best of our knowledge – well-supported by current contextual frameworks.

In terms of recurrent context, most examples stem from temporal context, where we may state something that recurrently holds true of a given date of a year, or a given day of the week, which we may then wish, for example, to map to intervals on a non-recurrent temporal context.

In terms of relative context, the Wikidata [67] knowledge graph defines a "truthy" context, which includes information that is not deprecated or superseded by other information; for example, a population reading for a city would be considered truthy if there is no other more recent population reading available. However, the contextual notion of "most recent" requires a relative assessment of context dependent on the other data available.

Regarding conceptual context, the triple $(\mathsf{Pluto}, \mathsf{type}, \mathsf{Dwarf\ Planet})$ only holds true since 2006. But unlike temporal context, it is the conceptualisation of a Planet, rather than Pluto itself, that has changed. This is an example of *concept drift* [70], where the meaning of domain terms can change over time, which in turn relates to the area of *ontology evolution* [75]. How we conceptualise a domain can thus also be contextual.

The notion of *context* in knowledge graphs can then be arbitrarily complex, where more complex notions of context remain poorly understood.

7 Embeddings

Machine learning has been gaining more and more attention in recent years, particularly due to impressive advances in sub-areas such as Deep Learning, where multi-layer architectures such as Convolutional Neural Networks (CNNs) have led to major strides in tasks involving multi-dimensional inputs (such as image classification), while architectures such as Recurrent Neural Networks (RNNs) have likewise lead to major strides in tasks involving sequential data (such as natural language translation). An obvious question then is: what kinds of learning architectures could be applied to graphs? The answer, unfortunately, is not so obvious. While machine learning architectures typically assume numeric inputs in the form of *tensors* (multi-dimensional arrays), graphs – unlike, say, images – are not naturally conceptualised in such terms.

A first attempt to encode a graph numerically would be a "one-hot encoding". Recalling the notation of N_G for the nodes of a del graph, we introduce a similar notation E_G to denote the edge-labels (i.e., predicates or properties) of a del graph; formally, $E_G \coloneqq \{p \mid \exists x, y : (x, p, y) \in G\}$. We could consider creating a 3-mode tensor of size $|N_G| \cdot |E_G| \cdot |N_G|$, where the element at (i, j, k) is 1 if $(n_i, e_j, n_k) \in G$, or 0 otherwise; here n_i and n_k denote the i^{th} and k^{th} nodes in an indexing of N_G, while e_j indicates the j^{th} edge label in an indexing of E_G. Though we now have a tensor representing the graph, for most graphs in practice, the tensor would be very sparse, with few 1's relative to 0's. As a result, using the tensor for the purposes of machine learning would be impractical.

In order to create more practical numeric representations of graphs for machine learning applications, knowledge graph embeddings aim to embed graphs within a low-dimensional, dense, numerical representation. There are many ways in which embeddings may be defined, but typically an embedding will map each node and each edge-label of a graph to an independent vector or matrix. For simplicity, we will assume embeddings that associate nodes and edge-labels to real-valued vectors of fixed dimension d, which we denote by the set $\mathbb{R}^d$.[5]

Definition 14 (Knowledge graph embedding). *Given a del graph G, a* knowledge graph embedding *of G is a pair of mappings (ε, ρ) such that $\varepsilon : N_G \to \mathbb{R}^d$ and $\rho : E_G \to \mathbb{R}^d$.*

Typically ε is known as an entity embedding, while ρ is known as a relation embedding. The knowledge graph embedding then consists of (typically low-dimensional) numeric representations of the node and edge-labels of the graph G, typically extracted such that the graph G can be (approximately) reconstructed from the embedding. Towards this goal, the graph can be conceptualised as a function $\gamma : \mathbf{C} \times \mathbf{C} \times \mathbf{C} \to \mathbb{R}_{[0,1]}$, where $\gamma(s, p, o) = 1$ if $(s, p, o) \in G$ or $\gamma(s, p, o) = 0$ if $(s, p, o) \notin G$. Instead of accepting the constants s, p, o, however, we could rather consider accepting the embeddings of those concepts: $\varepsilon(s)$, $\rho(p)$, $\varepsilon(o)$. This gives rise to the notion of a *plausibility scoring function.*

[5] In practice, knowledge graph embeddings can take complex-valued vectors, or real-valued matrices, or have entity and relation embeddings of different dimensions [68], and so forth, but such details are not exigent for our purposes.

Definition 15 (Plausibility). *A* plausibility scoring function *is a partial function $\phi : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{[0,1]}$. Given a del graph $G = (V, E, L)$, a triple $(s, p, o) \in N_G \times E_G \times N_G$, and a knowledge graph embedding (ε, ρ) of G, the plausibility of (s, p, o) is given as $\phi(\varepsilon(s), \rho(p), \varepsilon(o))$.*

Triples with scores closer to 1 are considered more plausible, while triples with scores closer to 0 are considered less plausible. We can now learn embeddings that yield a score close to 1 for triples in (s, p, o), and a score close to 0 for a sample of negative triples not in G. Given that G is assumed to be incomplete, we cannot be certain that triples not in G are false. A common heuristic to generate negative examples is to apply the Partial Completeness Assumption (PCA), taking a triple (s, p, o) from G and replacing a term (often o) with another term appearing in G such that the resulting triple does not appear in G. In practice, the dimensions of the embeddings are fixed to a low number such that rather than "remembering" G, the knowledge graph embedding is forced to generalise patterns in how G is connected. Different knowledge graph embeddings instantiate the types of embedding considered and the plausibility scoring function in different ways [68]. We refer to the extended tutorial for details [35].

Once trained over G, knowledge graph embeddings directly allow for estimating the plausibility for triples of the form $(s, p, o) \in N_G \times E_G \times N_G$ (i.e., triples using some combination of terms found in G). This can be used for the purposes of link prediction, whereby considering the graph G of Figure 1 and given a partial triple such as (Toliman, orbits, ?), the embedding of G can be used to predict likely completions of the triples, such as Rigil Kentaurus (already in the graph), or perhaps Proxima Centauri, or even Toliman itself. The embeddings themselves can also be useful independently of the plausibility scoring function as numerical representations of the elements of the graph, where, for example, similar numerical values will typically be assigned to similar nodes and edge-labels based on how they are connected in the graph.

In terms of open questions, a natural topic to explore is the combination of knowledge graph embeddings with ontologies and notions of entailment.

Topic 7.1: Semantic knowledge graph embeddings

The plausibility scoring function assigns 1 to triples deemed likely to be true, and 0 to triples deemed likely to be false. This function is learnt based on the triples found in G. But in the case that G contains ontological definitions, it may entail triples that are not explicitly stated. In such cases, many knowledge graph embeddings only consider the structure, rather than the semantics, of G, and may thus assign entailed-but-not-stated triples a score closer to 0. Ideally, a semantic knowledge graph embedding would assign a plausibility of (close to) 1 for triples that are entailed by G.

A natural strategy is to materialise as many entailments for G as possible and then apply a structural knowledge graph embedding over the extended version of G as usual. However, in the case of ontological definitions that generate new existentials, for example, it is possible that not all entailments can be materialised; in other cases, the number of entailments may simply be too large for materialisation to be practical.

A number of approaches have been proposed for considering entailments in knowledge graph embeddings [69, 28, 20]. Wang et al. [69] use functional and inverse-functional axioms as constraints (under a Unique Name Assumption (UNA)) such that if adding a triple to the graph would violate such a constraint, then its plausibility is lowered; for example, if we define system to be functional, and if we already have that the edge (Sun, system, Sol) holds, then we would reduce the plausibility of triples such as (Sun, system, Alpha Centauri). On the other hand, the KALE [28] system uses a combination of t-norm fuzzy logics and rules to assign plausibility to entailments. However, the approach relies on generating "ground rules" that indicate specific ways in which a triple can be entailed, which can lead to many ground rules for more complex definitions.

A perhaps simpler approach is to again conceptualise entailment numerically. If we state, for example, that `parent` $\sqsubseteq$ `orbits`, then we would expect that for any triples $(x,$ parent$, y)$ and $(x,$ orbits$, y)$, the former triple (being more specific) should be less plausible than the latter triple (being more general). Likewise, for example, a triple $(z,$ type, Dwarf Planet$)$ should always be less plausible than a triple $(z,$ type, Planemo$)$. This observation is exploited by FSL [20], which then defines soft constraints that associate a cost with contradicting such plausibility orderings. However, only simple forms of entailment are supported.

Hence a relevant topic is to explore how knowledge graph embeddings can be formulated and trained in order to find a numerical representation not only of the structure of a graph, but also its semantics, including potentially complex forms of entailment.

8 Graph Neural Networks

Rather than encoding the structure of graphs numerically, an alternative to enable learning over graphs is to design machine learning architectures that operate directly over the structure of a graph. A natural starting point is to consider neural networks, which already form graphs where nodes are (artificial) neurons, and edges are (artificial) axons that pass signals between neurons. Unlike graphs used to represent data, however, traditional neural networks tend to have a much more regular structure, being organised into layers, where all the nodes of one layer are connected pairwise to all the nodes of the next layer.

To enable learning over graphs of data, an alternative approach is to define the structure of the neural network in terms of the structure of the graph. In this case, the nodes of the graph are interpreted as neurons, while edges are interpreted as axons. Thus nodes can pass signals to each other through edges towards solving a given task. In a supervised setting, we may label a subset of nodes in the graph and then learn functions that aggregate information from the neighbourhood of each node, computing a new state for the node; this may continue recursively until a fixpoint, or for a fixed number of steps. Typically nodes and edges in the input graph will be associated with numerical feature vectors that encode pertinent information for the supervised task. Such a learning architecture is known as a graph neural network (GNN) [60].

As an example, assuming a large del graph similar to that shown in Figure 2, we may label a subset of planets that are known to be in a habitable zone that could support life. We could then associate nodes with vectors that numerically encode their type, their mass, their density, their composition etc.; we may then further associate edges with vectors indicating the type of relation, the astronomic distance between bodies, etc. Based on labelled examples, a graph neural network can then learn aggregation functions that take the information surrounding the neighbourhood of each node – for example, numerical data about the moon(s) of a particular planet, the star(s) of planet, the distances involved, etc. – and generate the expected output; the same functions can then be applied to unlabelled nodes to generate predicted classifications.

We first define a vector-labelled del graph, which serves as input to a GNN:

Definition 16 (Vector-labelled graph). *A* vector-labelled del graph G_λ *is a pair* $G_\lambda \coloneqq (G, \lambda)$ *where* G *is a del graph, and* $\lambda : N_G \cup G \to \mathbb{R}^a$ *is a vector labelling function.*

For simplicity, we will assume that nodes and triples are labelled with vectors of the same dimension (a). Thereafter, there are two principle architectures for GNNs: recursive and non-recursive graph neural networks. Here we focus on non-recursive graph neural networks, where details of the recursive architecture can rather be found in the extended tutorial [35]. Note that in the following definition, we use $S \to \mathbb{N}$ to denote *bags* (aka *multisets*) formed from the set S.

Definition 17 (Non-recursive graph neural network). *A* non-recursive graph neural network *(NRecGNN)* $\mathfrak{N}$ *with* l *layers is an* l*-tuple of functions* $\mathfrak{N} \coloneqq (\text{Agg}^{(1)}, \ldots, \text{Agg}^{(l)})$*, where* $\text{Agg}^{(k)} : \mathbb{R}^a \times 2^{(\mathbb{R}^a \times \mathbb{R}^a) \to \mathbb{N}} \to \mathbb{R}^a$ *for* $1 \leq k \leq l$*.*

Each aggregation function $\text{Agg}^{(k)}$ computes a new feature vector for a node, given its previous feature vector and the feature vectors of the nodes and edges forming its neighbourhood. We assume for simplicity that the dimensions of the vectors remain the same throughout, though this is not necessary in practice.[6] Given an NRecGNN $\mathfrak{N} = (\text{Agg}^{(1)}, \ldots, \text{Agg}^{(l)})$, a vector-labelled graph G_λ, and a node $u \in N_G$, we define the output vector assigned to node u in G_λ by $\mathfrak{N}$ (written $\mathfrak{N}(G_\lambda, u)$) as follows. First let $\mathbf{n}_u^{(0)} \coloneqq \lambda(u)$. For all $i \geq 1$, let:

$$\mathbf{n}_u^{(i)} \coloneqq \text{Agg}^{(i)}\left(\mathbf{n}_u^{(i-1)}, \{\!\{(\mathbf{n}_v^{(i-1)}, \lambda(v,p,u)) \mid (v,p,u) \in E\}\!\}\right)$$

Then $\mathfrak{N}(G_\lambda, u) \coloneqq \mathbf{n}_u^{(l)}$.

[6] We can still define a to be the largest dimension needed, padding other vectors.

In an l-layer NRecGNN, a different aggregation function is applied at each step (i.e., in each layer), up to a fixed number of steps l. These aggregation functions are typically parametrised and learnt based on labelled training data. When the aggregation functions are based on a convolutional operator, the result is a *convolutional graph neural network* (*ConvGNN*).

Though initially it may seem as if GNNs and ontologies are orthogonal concepts for graphs, there are some correspondences between both.

Topic 8.1

Both GNNs and ontologies can be used to classify nodes. While GNNs can be used to perform classification based on numerical methods and inductive learning, ontologies can be used to perform classification based on symbolic methods and deductive reasoning. An interesting research question is then to understand how these two paradigms might relate in terms of expressiveness. For example, can GNNs potentially learn to make similar classifications based on similar input data as ontologies? In fact, some progress has been made recently on this question.

GNNs are typically based on aggregations of data from a local neighbourhood. This means that there are limitations to what GNNs can distinguish. In particular, GNNs based on local information cannot distinguish certain non-isomorphic graphs [74], but rather can only distinguish graphs that are also distinguishable by a weaker *Weisfeiler–Lehman* (*WL*) test for isomorphism, which involves recursively hashing neighbouring information of nodes up to a fixpoint. More specifically, if the WL test assigns the same hash to two nodes in the graph, then those nodes cannot be distinguished by a GNN of the form previously described. Barceló et al. [10] then recently showed that for any binary classification expressible as an $\mathcal{ALCQ}$ ontology, there exists a GNN of the form previously described that will compute the same classification; they further define a discrete form of GNN that can only capture binary classifications expressible as an $\mathcal{ALCQ}$ ontology.

This rather surprising correspondence between GNNs and $\mathcal{ALCQ}$ constitutes a bridge between deductive and inductive semantics for knowledge graphs. An interesting line of research is then to investigate GNN-style architectures that permit classification with respect to other DLs [10].

9 Conclusions

The growing popularity of knowledge graphs presents an opportunity for research that combines various technical perspectives on how graphs can be used to represent and exploit knowledge at large scale. Within the intersection of these varying perspectives lies interesting research questions in terms of how concepts from graph databases and knowledge representation can be fruitfully combined, how techniques from machine learning relate to logical languages, and so forth.

Here we have mentioned a number of key concepts relating to knowledge graphs – specifically data models, querying, ontologies, rules, embeddings and graph neural networks – as well as a number of research topics that arise from their combination. We have only scratched the surface. Knowledge graphs encompass an even wider range of topics, where we have not touched upon concepts such as shapes and validation, graph algorithm and analytics, privacy and anonymisation, data quality, knowledge graph creation and completion, knowledge graph refinement, etc. Likewise there are many interesting research questions that arise when considering combinations of these concepts. For discussion on these and other topics we refer to the extended tutorial [35].

Acknowledgements This work was supported by Fondecyt Grant No. 1181896 and by the Millennium Institute for Foundational Research on Data (IMFD). I would like to thank my co-authors on the extended tutorial for the various discussions and contributions that helped to inform these lecture notes. I also wish to thank the anonymous reviewers' for their helpful comments.

References

[1] Aberger, C.R., Lamb, A., Tu, S., Nötzli, A., Olukotun, K., Ré, C.: Emptyheaded: A relational engine for graph processing. ACM Transactions on Database Systems (TODS) 42(4), 20 (2017)
[2] Angles, R.: The Property Graph Database Model. In: Olteanu, D., Poblete, B. (eds.) Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21–25, 2018. CEUR Workshop Proceedings, vol. 2100. Sun SITE Central Europe (CEUR) (2018), http://ceur-ws.org/Vol-2100/paper26.pdf
[3] Angles, R., Arenas, M., Barceló, P., Boncz, P.A., Fletcher, G.H.L., Gutierrez, C., Lindaaker, T., Paradies, M., Plantikow, S., Sequeda, J.F., van Rest, O., Voigt, H.: G-CORE: A Core for Future Graph Query Languages. In: [19], pp. 1421–1432
[4] Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of Modern Query Languages for Graph Databases. ACM Computing Surveys 50(5), 68:1–68:40 (2017)
[5] Arenas, M., Conca, S., Pérez, J.: Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In: Mille, A., Gandon, F.L., Misselis, J., Rabinovich, M., Staab, S. (eds.) Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012. pp. 629–638. ACM Press (Apr 2012)
[6] Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite Family and Relations. Journal of Artificial Intelligence Research 36, 1–69 (2009)
[7] Atserias, A., Grohe, M., Marx, D.: Size bounds and query plans for relational joins. SIAM J. Comput. 42(4), 1737–1767 (2013), https://doi.org/10.1137/110859440

[8] Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press, Cambridge, United Kingdom (2017)

[9] Barceló, P.: Querying graph databases. In: Hull, R., Fan, W. (eds.) Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013. pp. 175–188. ACM Press (Jun 2013), https://doi.org/10.1145/2463664.2465216

[10] Barceló, P., Kostylev, E.V., Monet, M., Peréz, J., Reutter, J., Silva, J.P.: The Logical Expressiveness of Graph Neural Networks. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020. OpenReview.net (Apr 2020), https://openreview.net/forum?id=r1lZ7AEKvB

[11] Bellomarini, L., Sallinger, E., Gottlob, G.: The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. Proceedings of the VLDB Endowment 11(9), 975–987 (2018)

[12] Bienvenu, M., Ortiz, M., Simkus, M.: Regular Path Queries in Lightweight Description Logics: Complexity and Algorithms. J. Artif. Intell. Res. 53, 315–374 (2015)

[13] Bischof, S., Krötzsch, M., Polleres, A., Rudolph, S.: Schema-Agnostic Query Rewriting in SPARQL 1.1. In: [47], pp. 584–600

[14] Bollacker, K., Tufts, P., Pierce, T., Cook, R.: A platform for scalable, collaborative, structured information integration. In: Nambiar, U., Nie, Z. (eds.) Intl. Workshop on Information Integration on the Web (IIWeb'07) (2007)

[15] Bonifati, A., Martens, W., Timm, T.: An Analytical Study of Large SPARQL Query Logs. Proceedings of the VLDB Endowment 11(2), 149–161 (2017)

[16] Capadisli, S., Auer, S., Ngomo, A.N.: Linked SDMX Data: Path to high fidelity Statistical Linked Data. Semantic Web 6(2), 105–112 (2015)

[17] Carral, D., Dragoste, I., González, L., Jacobs, C.J.H., Krötzsch, M., Urbani, J.: VLog: A Rule Engine for Knowledge Graphs. In: [25], pp. 19–35

[18] Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014. W3c recommendation, World Wide Web Consortium (Feb 25 2014), https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[19] Das, G., Jermaine, C.M., Bernstein, P.A. (eds.): Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018. ACM Press (Jun 2018)

[20] Demeester, T., Rocktäschel, T., Riedel, S.: Lifted Rule Injection for Relation Embeddings. In: [66], pp. 1389–1399

[21] Dimartino, M.M., Calì, A., Poulovassilis, A., Wood, P.T.: Efficient Ontological Query Answering by Rewriting into Graph Queries. Lecture Notes in Computer Science, vol. 11529, pp. 75–84. Springer (2019)

[22] Feier, C., Kuusisto, A., Lutz, C.: Rewritability in Monadic Disjunctive Datalog, MMSNP, and Expressive Description Logics. Log. Methods Comput. Sci. 15(2) (2019)

[23] Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., Taylor, A.: Cypher: An Evolving Query Language for Property Graphs. In: [19], pp. 1433–1445

[24] Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: Fast rule mining in ontological knowledge bases with AMIE+. The Very Large Data Base Journal 24(6), 707–730 (2015)

[25] Ghidini, C., Hartig, O., Maleshkova, M., Svátek, V., Cruz, I.F., Hogan, A., Song, J., Lefrançois, M., Gandon, F. (eds.): The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II, Lecture Notes in Computer Science, vol. 11779. Springer (Oct 2019)

[26] Gottlob, G., Orsi, G., Pieris, A., Simkus, M.: Datalog and Its Extensions for Semantic Web Databases. Lecture Notes in Computer Science, vol. 7487, pp. 54–77. Springer (2012)

[27] Grau, B.C., Motik, B., Stoilos, G., Horrocks, I.: Computing Datalog Rewritings Beyond Horn Ontologies. In: Rossi, F. (ed.) IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013. pp. 832–838. IJCAI/AAAI (Aug 2013)

[28] Guo, S., Wang, Q., Wang, L., Wang, B., Guo, L.: Jointly Embedding Knowledge Graphs and Logical Rules. In: [66], pp. 192–202

[29] Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013. W3c recommendation, World Wide Web Consortium (Mar 21 2013), `https://www.w3.org/TR/2013/REC-sparql11-query-20130321/`

[30] Hartig, O.: Foundations of RDF* and SPARQL* – An Alternative Approach to Statement-Level Metadata in RDF. In: Reutter, J.L., Srivastava, D. (eds.) Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017. CEUR Workshop Proceedings, vol. 1912. Sun SITE Central Europe (CEUR) (2017), `http://ceur-ws.org/Vol-1912/paper12.pdf`

[31] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S.: OWL 2 Web Ontology Language Primer (Second Edition), W3C Recommendation 11 December 2012. W3c recommendation, World Wide Web Consortium (Dec 11 2012), `https://www.w3.org/TR/2012/REC-owl2-primer-20121211/`

[32] Ho, V.T., Stepanova, D., Gad-Elrab, M.H., Kharlamov, E., Weikum, G.: Rule Learning from Knowledge Graphs Guided by Embedding Models. In: Vrandečić, D., Bontcheva, K., Suárez-Figueroa, M.C., Presutti, V., Celino, I., Sabou, M., Kaffee, L., Simperl, E. (eds.) The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11136, pp. 72–90. Springer (Oct 2018)

[33] Hoffart, J., Suchanek, F.M., Berberich, K., Lewis-Kelham, E., de Melo, G., Weikum, G.: YAGO2: Exploring and querying world knowledge in time, space, context, and many languages. In: Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M.P., Bertino, E., Kumar, R. (eds.) Proceedings

of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011 (Companion Volume). pp. 229–232. ACM Press (Mar 2011)

[34] Hogan, A., Arenas, M., Mallea, A., Polleres, A.: Everything you always wanted to know about blank nodes. Journal of Web Semantics 27–28, 42–69 (2014)

[35] Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutierrez, C., Gayo, J.E.L., Kirrane, S., Neumaier, S., Polleres, A., Navigli, R., Ngomo, A.N., Rashid, S.M., Rula, A., Schmelzeisen, L., Sequeda, J.F., Staab, S., Zimmermann, A.: Knowledge graphs. CoRR abs/2003.02320 (2020), `https://arxiv.org/abs/2003.02320`

[36] Hogan, A., Riveros, C., Rojas, C., Soto, A.: A worst-case optimal join algorithm for SPARQL. In: Ghidini, C., Hartig, O., Maleshkova, M., Svátek, V., Cruz, I.F., Hogan, A., Song, J., Lefrançois, M., Gandon, F. (eds.) The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11778, pp. 258–275. Springer (Oct 2019)

[37] Homola, M., Serafini, L.: Contextualized Knowledge Repositories for the Semantic Web. Journal of Web Semantics 12, 64–87 (2012)

[38] Kalinsky, O., Etsion, Y., Kimelfeld, B.: Flexible Caching in Trie Joins. In: International Conference on Extending Database Technology (EDBT). pp. 282–293. OpenProceedings.org (2017)

[39] Krötzsch, M., Marx, M., Ozaki, A., Thost, V.: Attributed Description Logics: Reasoning on Knowledge Graphs. pp. 5309–5313 (IJCAI:2018), `https://doi.org/10.24963/ijcai.2018/743`

[40] Krötzsch, M., Rudolph, S., Schmitt, P.H.: A closer look at the semantic relationship between Datalog and description logics. Semantic Web 6(1), 63–79 (2015)

[41] Krötzsch, M., Simancik, F., Horrocks, I.: Description Logics. IEEE Intelligent Systems 29(1), 12–19 (2014)

[42] LaPaugh, A.S., Papadimitriou, C.H.: The even-path problem for graphs and digraphs. Networks 14(4), 507–513 (1984), `https://doi.org/10.1002/net.3230140403`

[43] Lefrançois, M., Zimmermann, A.: The Unified Code for Units of Measure in RDF: cdt: ucum and other UCUM Datatypes. In: Gangemi, A., Gentile, A.L., Nuzzolese, A.G., Rudolph, S., Maleshkova, M., Paulheim, H., Pan, J.Z., Alam, M. (eds.) The Semantic Web: ESWC 2018 Satellite Events - ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11155, pp. 196–201. Springer (Jun 2018)

[44] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web Journal 6(2), 167–195 (2015)

[45] Libkin, L.: Locality of queries and transformations. Electron. Notes Theor. Comput. Sci. 143, 115–127 (2006), https://doi.org/10.1016/j.entcs.2005.04.041

[46] Meroño-Peñuela, A., Daga, E.: List.MID: A MIDI-Based Benchmark for Evaluating RDF Lists. In: [25], pp. 246–260

[47] Mika, P., Tudorache, T., Bernstein, A., Welty, C.A., Knoblock, C.A., Vrandecic, D., Groth, P.T., Fridman Noy, N., Janowicz, K., Goble, C.A. (eds.): The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I, Lecture Notes in Computer Science, vol. 8796. Springer (Oct 2014)

[48] Miller, J.J.: Graph Database Applications and Concepts with Neo4j. In: Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA March 23rd-24th, 2013. pp. 141–147. AIS eLibrary (2013), https://aisel.aisnet.org/sais2013/24

[49] Navigli, R., Ponzetto, S.P.: BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artificial Intelligence 193, 217–250 (2012)

[50] Ngo, H.Q., Porat, E., Ré, C., Rudra, A.: Worst-case optimal join algorithms. J. ACM 65(3), 16:1–16:40 (2018), https://doi.org/10.1145/3180143

[51] Nguyen, D., Aref, M., Bravenboer, M., Kollias, G., Ngo, H.Q., Ré, C., Rudra, A.: Join processing for graph patterns: An old dog with new tricks. In: GRADES. p. 2. ACM (2015)

[52] Nguyen, V., Bodenreider, O., Sheth, A.: Don't Like RDF Reification?: Making Statements About Statements Using Singleton Property. In: Chung, C.W., Broder, A.Z., Shim, K., Suel, T. (eds.) 23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014. pp. 759–770. ACM Press (Apr 2014)

[53] Noy, N.F., Gao, Y., Jain, A., Narayanan, A., Patterson, A., Taylor, J.: Industry-scale Knowledge Graphs: Lessons and Challenges. ACM Queue 17(2), 20 (2019)

[54] Ortiz, M., Rudolph, S., Simkus, M.: Query Answering in the Horn Fragments of the Description Logics SHOIQ and SROIQ. In: Walsh, T. (ed.) IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011. pp. 1039–1044. IJCAI/AAAI (Aug 2011)

[55] Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P.F. (eds.): Reasoning Web. Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures, Lecture Notes in Computer Science, vol. 6848. Springer (Aug 2011)

[56] Reutter, J.L., Soto, A., Vrgoc, D.: Recursion in SPARQL. In: Arenas, M., Corcho, Ó., Simperl, E.P.B., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P.T., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (eds.) The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I.

Lecture Notes in Computer Science, vol. 9366, pp. 19–35. Springer (Oct 2015)

[57] Rodriguez, M.A.: The Gremlin graph traversal machine and language. In: Cheney, J., Neumann, T. (eds.) Proceedings of the 15th Symposium on Database Programming Languages, Pittsburgh, PA, USA, October 25-30, 2015. pp. 1–10. ACM Press (Oct 2015)

[58] Rudolph, S.: Foundations of Description Logics. In: [55], pp. 76–136

[59] Rudolph, S., Krötzsch, M., Hitzler, P.: Description Logic Reasoning with Decision Diagrams: Compiling SHIQ to Disjunctive Datalog. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T.W., Thirunarayan, K. (eds.) The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5318, pp. 435–450. Springer (2008)

[60] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The Graph Neural Network Model. IEEE Transactions on Neural Networks 20(1), 61–80 (2009)

[61] Schmidt-Schauß, M., Smolka, G.: Attributive Concept Descriptions with Complements. Artificial Intelligence 48(1), 1–26 (1991)

[62] Schuetz, C., Bozzato, L., Neumayr, B., Schrefl, M., Serafini, L.: Knowledge Graph OLAP: A Multidimensional Model and Query Operations for Contextualized Knowledge Graphs. Semantic Web Journal (2020), (Under open review)

[63] Sequeda, J.F., Arenas, M., Miranker, D.P.: OBDA: Query Rewriting or Materialization? In Practice, Both! In: [47], pp. 535–551

[64] Singhal, A.: Introducing the Knowledge Graph: things, not strings. Google Blog (May 2012), `https://www.blog.google/products/search/introducing-knowledge-graph-things-not/`

[65] Stefanoni, G., Motik, B., Krötzsch, M., Rudolph, S.: The Complexity of Answering Conjunctive and Navigational Queries over OWL 2 EL Knowledge Bases. J. Artif. Intell. Res. 51, 645–705 (2014)

[66] Su, J., Carreras, X., Duh, K. (eds.): Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016. The Association for Computational Linguistics (Nov 2016)

[67] Vrandečić, D., Krötzsch, M.: Wikidata: A Free Collaborative Knowledgebase. Communications of the ACM 57(10), 78–85 (2014)

[68] Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge Graph Embedding: A Survey of Approaches and Applications. IEEE Transactions on Knowledge and Data Engineering 29(12), 2724–2743 (Dec 2017)

[69] Wang, Q., Wang, B., Guo, L.: Knowledge Base Completion Using Embeddings and Rules. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 1859–1866. IJCAI/AAAI (Jul 2015)

[70] Wang, S., Schlobach, S., Klein, M.C.A.: Concept drift and how to identify it. J. Web Semant. 9(3), 247–265 (2011)

[71] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A Comprehensive Survey on Graph Neural Networks. CoRR abs/1901.00596 (2019), `http://arxiv.org/abs/1901.00596`

[72] Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-Based Data Access: A Survey. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. pp. 5511–5519. IJCAI/AAAI (Jul 2018)

[73] Xiao, G., Rezk, M., Rodriguez-Muro, M., Calvanese, D.: Rules and Ontology Based Data Access. In: Web Reasoning and Rule Systems - 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings. pp. 157–172 (2014), `https://doi.org/10.1007/978-3-319-11113-1_11`

[74] Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How Powerful are Graph Neural Networks? In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (May 2019), `https://openreview.net/forum?id=ryGs6iA5Km`

[75] Zablith, F., Antoniou, G., d'Aquin, M., Flouris, G., Kondylakis, H., Motta, E., Plexousakis, D., Sabou, M.: Ontology evolution: a process-centric survey. Knowledge Eng. Review 30(1), 45–75 (2015)

[76] Zimmermann, A., Lopes, N., Polleres, A., Straccia, U.: A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data. Journal of Web Semantics 12, 72–95 (mar 2012)