UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Data Science Curriculum

Mart Traagel

Comparative Analysis of Deterministic and Graph Neural Network Based RDFS Materialization Methods

Master's Thesis (15 ECTS)

Supervisor:
Bruno Rucy Carneiro Alves de Lima

Tartu 2023

Comparative Analysis of Deterministic and Graph Neural Network Based RDFS Materialization Methods

Abstract:
This thesis compares deterministic Datalog-based and modern deep learning-based methodologies for Resource Description Framework Schema (RDFS) materialization. The research process was meticulously divided into distinct stages. The central focus was to examine the performance of the two methods regarding efficiency and effectiveness. The results indicated that while deep learning approaches, particularly Graph Neural Networks, demonstrated the capability to handle complex graph-structured data, they were considerably slower than their Datalog counterparts. These findings illuminate both methodologies' strengths and limitations, providing crucial insights for future exploration in this domain.

Keywords: RDF, RDFS, Graph Neural Networks, Knowledge Graph Materialization, Neural Network, Machine Learning

CERCS: P176 - Artificial intelligence

Deterministlike ja Graaf-Närvivõrgu Põhiste RDFS-i Materialiseerimismeetodite Võrdlev Analüüs

Lühikokkuvõte:
Käesolev lõputöö võrdleb deterministlikke Datalogi põhiseid ja kaasaegseid süvaõppe-põhiseid meetodeid *Resource Description Framework Schema* (RDFS) materialiseerimiseks. Uurimisprotsess jagati hoolikalt eraldi etappideks. Keskne fookus oli uurida kahe meetodi efektiivsust ja tõhusust. Tulemused näitasid, et kuigi süvaõppe meetodid, eriti graaf-närvivõrkude meetodid, demonstreerisid võimet käsitleda keerukat graafistruktuuriga andmeid, olid nad oluliselt aeglasemad kui Datalogi meetodid. Antud tulemused valgustavad mõlema metoodika tugevusi ja piiranguid, pakkudes olulisi teadmisi tulevaseks uurimistööks selles valdkonnas.

Võtmesõnad: RDF, RDFS, Graafnärvivõrgud, Närvivõrk, Masinõpe

CERCS: P176 - Tehisintellekt

COMPARATIVE ANALYSIS OF TRADITIONAL AND GRAPH NEURAL NETWORK-BASED RDFS MATERIALIZATION METHODS

Data Science (MSc), 2023
Mart Traagel
Tartu University, Institute of Computer Science

This master's thesis investigates the performance of neural network-based inference in comparison to traditional querying methods for RDFS materialization, with a focus on evaluating efficiency gains.

Abstract

This thesis compares deterministic Datalog-based and modern deep learning-based methodologies for Resource Description Framework Schema (RDFS) materialization.

The research process was meticulously divided into distinct stages. The central focus was to examine the performance of the two methods regarding efficiency and effectiveness.

The results indicated that while deep learning approaches, particularly Graph Neural Networks, demonstrated the capability to handle complex graph-structured data, they were considerably slower than their Datalog counterparts.

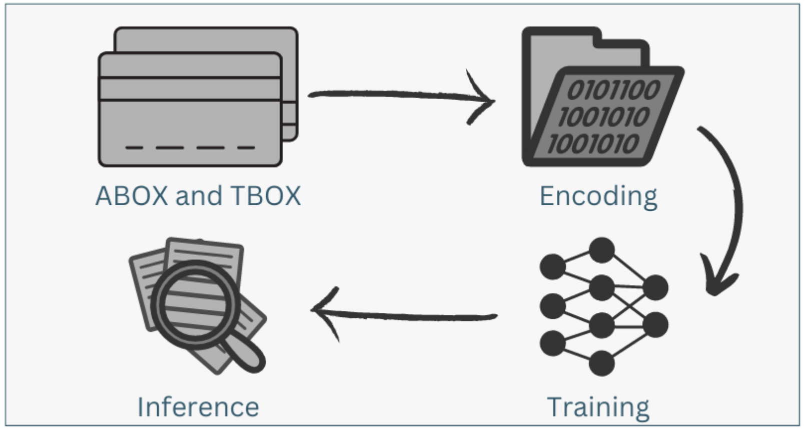
These findings illuminate both methodologies' strengths and limitations, providing crucial insights for future exploration in this domain.

Steps

1. The user provides a set of ground truth rules
2. Encoding the data
3. Training the model
4. Run inference in place of querying

Results

In assessing the inference process, a single graph took between 1 to 30 seconds, a timeframe that becomes problematic when handling larger datasets of over 20,000 graphs. The graph reasoner, therefore, lagged significantly in efficiency compared to its Datalog counterpart.



Procedure

The ABOX (Assertional Box) contains the ground truths, the specific facts or instances about the domain, while the TBOX (Terminological Box) contains the schema or model, which consists of the ontology's classes, properties, and relationships. By applying the rules of TBOX on the assertions in the ABOX, we can infer new information. By inferring all possible information we materialize the knowledge graph.

Conclusion

The differential efficiency between the two underlines the challenges that persist in optimizing the performance of graph reasoners for extensive, complex datasets, and emphasizes the ongoing relevance of traditional Datalog reasoning methods in certain contexts.

DETERMINISTLIKE JA GRAAF-NÄRVIVÕRGU PÕHISTE RDFS-I MATERIALISEERIMISMEETODITE VÕRDLEV ANALÜÜS

Andmeteadus (MSc), 2023
Mart Traagel
Tartu Ülikool, Arvutiteaduste Instituut

Käesolev töö võrdleb traditsiooniliste deterministlike materialiseerimismeetodeid graaf-närvivõrgu põhiste meetoditega eesmärgiga hinnata mõlema meetodi tõhusust.

Lühikokkuvõte

Antud töö võrdleb deterministlikke Datalogi põhiseid ja kaasaegseid süvaõppe-põhiseid meetodeid *Resource Description Framework Schema* (RDFS) materialiseerimiseks.

Uurimisprotsess jagati hoolikalt eraldi etappideks. Keskne fookus oli uurida kahe meetodi efektiivsust ja tõhusust. Tulemused näitasid, et kuigi süvaõppe meetodid, eriti graaf-närvivõrkude meetodid, demonstreerisid võimet käsitleda keerukat graafistruktuuriga andmeid, olid nad oluliselt aeglasemad kui Datalogi meetodid.

Antud tulemused valgustavad mõlema metoodika tugevusi ja piiranguid, pakkudes olulisi teadmisi tulevaseks uurimistööks selles valdkonnas.

Sammud

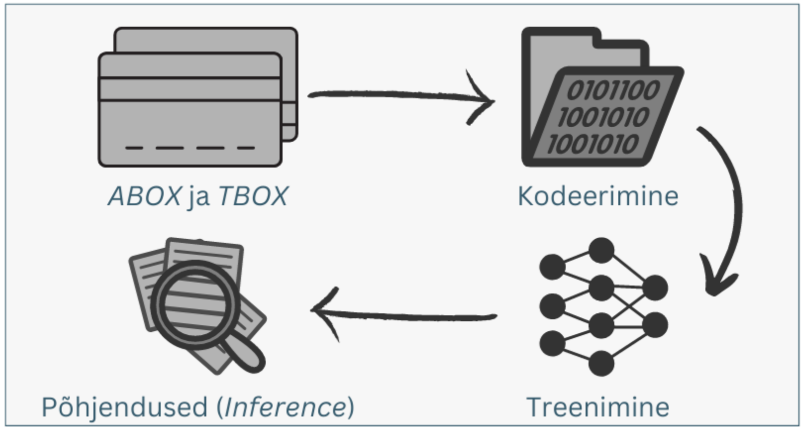
1. Kasutaja esitab põhitõdede reeglite kogumi
2. Andmete kodeerimine
3. Mudeli treenimine
4. *Päringu* asemel kasutatakse *järeldamist*

Protsessi kirjeldus

ABOX (*Assertional Box*) sisaldab põhitõdesid, mis on domeeni kohta käivad spetsiifilised faktid või näited, samas kui TBOX (Terminological Box) sisaldab skeemi või mudelit, mis koosneb ontoloogia klassidest, omadustest ja suhetest. TBOXi reeglite rakendamine ABOXis esitatud väidetele võimaldab meil teha uusi järeldusi. Kõigi võimalike andmete tuletamise teel *materialiseerime* teadmiste graafi.

Tulemused

Järeldusprotsessi hindamisel võttis ühe graafiku jaoks aega 1 kuni 30 sekundit, ajavahemik, mis on probleem suurtel andmekogustel mis võivad koosnevad üle 20 000 graafist. Seega jäi süvaõppe meetodi *põhjendaja* efektiivsuses märkimisväärselt alla Datalogi *põhjendajale*.



Kokkuvõte

Kahe vahel esinev erinevus efektiivsuses rõhutab probleeme, mis püsivad graafide põhjendajate jõudluse optimeerimisel ulatuslike ja keerukate andmekogumite jaoks ning rõhutab traditsiooniliste deterministlike Datalogi põhjendusmeetodite püsivat olulisust teatud kontekstides.

Table of Contents

Acknowledgements

I would like to thank my cat Yolandi for keeping me company through the course of this thesis. I am also very grateful to my supervisor Rucy for opening the door for me to the world of knowledge graphs, ontologies, and reasoners. During my Master's studies, I have learned a lot, and even more, I have discovered how much there is still to know. As I consider learning one of the highest virtues, and a privilege, I owe my thanks to everyone who contributes to the persistence and maintenance of the institutes that allow learning to continue.

Methodology

Various advanced Artificial Intelligence based tools were utilized in the development process of this thesis. Firstly to enhance the quality and readability of the text. One such tool was ChatGPT, a state-of-the-art language model developed by OpenAI, which was instrumental in several aspects of the paper's preparation. Grammarly, an AI-powered writing assistant, was another essential tool to enhance the text's grammatical correctness and overall readability. It provided real-time grammar and spelling checks, enabling the author to refine the language and structure of the paper.

ChatGPT was also used to generate snippets of code, such as the code necessary for generating LaTeX graphs from a set of triples. It was also utilized to explain and comment on complex sections of code. For code writing and comprehension, GitHub Copilot was leveraged. GitHub Copilot is an AI-powered code completion tool that assists in writing code and understanding the logic behind existing code snippets.

Additionally, the auto-suggestion features of Google Docs were employed to streamline the writing process, making it faster and more efficient. The combined use of these tools significantly contributed to the quality and clarity of the research paper, ensuring that it was technically sound and accessible, and engaging for the reader.

The primary programming language employed in this research was Python, owing to its versatility and robustness in handling data analysis and machine learning tasks. PyCharm, an integrated development environment (IDE) by JetBrains, was utilized for the development environment. This IDE offers a comprehensive toolkit for Python programming, supporting various libraries and frameworks, making it a good choice for this topic. Importantly, the University of Tartu generously provided a student license for PyCharm, facilitating the use of this sophisticated tool and thereby contributing to the efficacy of the research process.

In addition to the resources above, this study sought to leverage the High-Performance Computing Centre (UTHPC) provided by the University of Tartu. Unfortunately, UTHPC utilization for experiments did not yield the anticipated productive outcomes. Despite this, the availability and potential of such a resource underscores the opportunities for future research and exploration in similar domains.

Introduction

RDFS (RDF Schema) materialization is the process of computing the complete RDFS closure or inferences of an RDF graph. This process includes applying the RDFS rules, such as subclass and property inheritance, inferring new triples (subject - predicate - object statements) that are logically implied by the schema and input graph [1], [2]. Materialization is a core aspect of RDF and RDFS reasoning - completing the graph and making it consistent - enabling more expressive and efficient querying.

Since RDF and RDFS are foundational technologies for the Semantic Web, materialization as a technique unlocks the full potential of linked data, enabling functional reasoning capabilities over large and interconnected datasets [2]. Precomputing the relevant inferences facilitates more efficient data integration, knowledge discovery, and interoperability [3]. There are several methods for performing materialization, each with strengths and weaknesses. Comparing available approaches will provide valuable insight into their relative performance, scalability, and applicability. The methods compared in this thesis use Datalog and a graph-based deep learning method [2].

For example, for given assertions for a generated sub-graph from the Lehigh University Benchmark (LUBM) [4] ontology:

G = *<http://www.Department0.University0.edu/AssistantProfessor0/**Publication1**>*

An N-triple representation of the graph G is:

Subject: *<http://www.Department0.University0.edu/AssistantProfessor0/**Publication1**>*
Predicate: *<http://www.w3.org/1999/02/22-rdf-syntax-ns#**type**>*
Object: *<http://swat.cse.lehigh.edu/onto/univ-bench.owl#**Publication**> .*

Subject: *<http://www.Department0.University0.edu/AssistantProfessor0/**Publication1**>*
Predicate: *<http://swat.cse.lehigh.edu/onto/univ-bench.owl#**publicationAuthor**>*
Object: *<http://www.Department0.University0.edu/**AssistantProfessor0**> .*

Subject: *<http://www.Department0.University0.edu/AssistantProfessor0/**Publication1**>*
Predicate: *<http://swat.cse.lehigh.edu/onto/univ-bench.owl#**name**>*
Object: *"**Publication1**" .*

This graph G can also be represented visually.

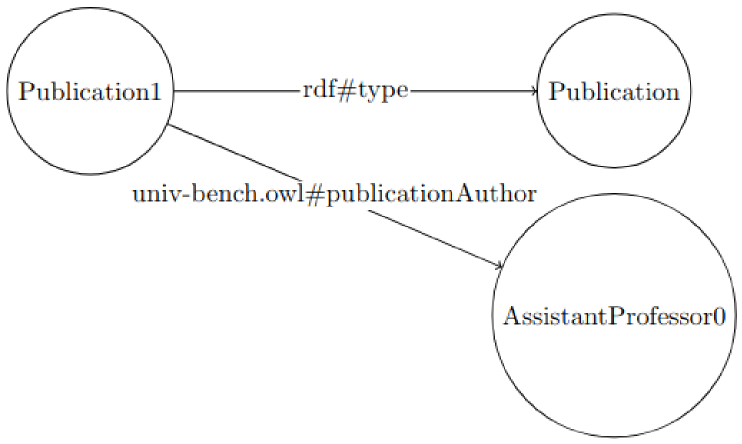


Figure 1: Visual representation of sub-graph G.

This information could be interpreted as follows: A publication, denoted as *'Publication1'*, has been authored by an entity titled *'AssistantProfessor0'*. While materializing the graph by applying ontology-specific terminological rules, it is revealed that *'Publication1'* also falls under the RDFS classification of *'Resource'*. Similarly, *'AssistantProfessor0'* is identified as an entity of the *'Resource'* type and, additionally, of the *'Person'* type. This underlines the capacity of semantic rules to infer and materialize unseen yet logical relationships within the data structure.

The N-triple representation is:
Subject: <*http://www.Department0.University0.edu/**AssistantProfessor0/Publication1***>
Predicate: <*http://www.w3.org/1999/02/22-rdf-syntax-ns#**type***>
Object: <*http://www.w3.org/2000/01/rdf-schema#**Resource***> .

Subject: <*http://www.Department0.University0.edu/**AssistantProfessor0***>
Predicate: <*http://www.w3.org/1999/02/22-rdf-syntax-ns#**type***>
Object: <*http://www.w3.org/2000/01/rdf-schema#**Resource***> .

Subject: <*http://www.Department0.University0.edu/**AssistantProfessor0***>
Predicate: <*http://www.w3.org/1999/02/22-rdf-syntax-ns#**type***>
Object: <*http://swat.cse.lehigh.edu/onto/univ-bench.owl#**Person***> .

This sub-graph can now be considered materialized, as inferred information has been added. The graph can be represented visually.

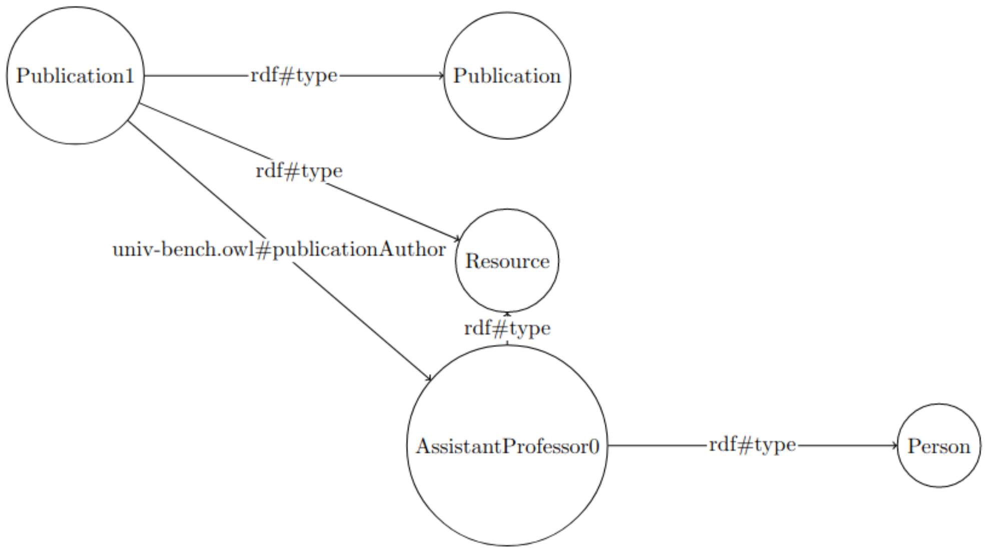


Figure 2. Visual representation of materialized sub-graph G.

Applying the logic rules across the entire ontology and saving all the inferred information in a database concludes the process of materialization, and any queries run on the database can use this already existing information.

Datalog is a declarative query language based on symbolic reasoning [5], offering several advantages in materialization [6]. It can naturally represent RDFS vocabulary and semantics as Datalog predicates and clauses, allowing for efficient reasoning over RDF data. The available methods are highly expressive. This enables formulation of complex rules and constraints and a declarative and human-readable representation. [5]

Machine learning methods, specifically those based on graph neural networks (GNNs), are an alternative method for RDFS materialization. These methods can learn from graph-structured data, capturing local and global information from the graph [7], [8]. Machine learning methods can adapt to the specific characteristics of a given RDF graph and RDFS schema, potentially leading to better performance and generalization across diverse datasets [9]. Machine learning methods can also utilize advances in hardware and software, such as GPU acceleration and distributed computing, to scale up materialization tasks for large and dynamic graphs [7].

A comparison between these methods aims to determine the strengths and weaknesses of each approach and provide insights into their suitability for different use cases, domains, and datasets, contributing to developing more efficient, scalable, and expressive techniques for RDFS materialization. This objective has been stated as two research questions:

1. How do the Datalog-based and machine learning-based methods perform regarding the materialized RDF graphs' accuracy, completeness, and consistency [10]?
2. How do the Datalog-based and machine learning-based methods compare regarding computational efficiency, memory requirements, and scalability [7]?

This thesis addresses these two questions, guiding the choice of appropriate methods for RDFS reasoning techniques in the Semantic Web. As there are many possible variations and implementations of these methods, the research will focus on representative techniques that showcase the key characteristics of both approaches. This limitation is imposed due to the vast array of techniques available in both Datalog-based and machine learning-based methods [6]–[9], which makes it infeasible to compare every possible combination.

The comparison of Datalog-based and machine learning-based methods will be conducted using a set of benchmark datasets - The Lehigh University Benchmark (LUBM) [4] and evaluation metrics (accuracy, precision, confusion matrices). These benchmarks and metrics will not cover all possible scenarios, domains, or use cases. Still, they will be chosen to represent a range of RDF graphs and schemas. The results obtained using these benchmarks and metrics may not be directly generalizable to other datasets or scenarios due to the inherent noise because the Web of data is inherently noisy [2].

Another limitation of the scope of this thesis is using hybrid approaches - not materializing the whole graph in a specific method but using it where it is most efficient. Instead, the research will focus on identifying the strengths of both methods and addressing their limitations, paving the way for future work in this area. This thesis aims to contribute to a better understanding of the qualities and shortcomings, assisting in choosing a suitable strategy for different RDF and RDFS reasoning assignments.

1. Background

1.1. Reasoning

1.1.1. Datalog

Datalog, a query language derived from Prolog, is grounded in first-order logic and Horn clauses. Originating from the inception of logic programming, it is commonly utilized to describe systems or construct domain models. This flexible language can query data from various systems, model relational data, and generate new data models with minimal code [11] [5].

Datalog is compatible with the Extensible Data Notation (edn), a data format used in programming languages like Clojure, and has been implemented in XTDB and datascript. It can query data from various sources, including relational structures, graphs, XML files, text files, and in-memory data structures. Notably, Datalog serves as the primary query language for the Datomic database [11].

Datalog's unique combination of simplicity and expressiveness renders it particularly well-suited for tackling complex queries and recursive rules on large datasets. This has contributed to its widespread adoption and popularity across various applications [12]. Its efficiency and optimization capabilities have increased usage in various settings, from database management systems and knowledge representation to deductive databases and artificial intelligence [11], [12].

Datalog has proven to be an invaluable tool in the context of database management systems. It empowers users to express complex queries, constraints, and views on relational data, enabling greater understanding and control over the data. Moreover, deductive databases have also reaped the benefits of Datalog's expressiveness by employing it for rule-based reasoning and query optimization. This has resulted in more efficient and effective operations within these databases, showcasing the versatility and strength of Datalog as a language. [11]

As a knowledge representation tool, Datalog has demonstrated its capacity to define ontologies, rules, and axioms that accurately capture the semantics of various domains. This is particularly beneficial in the domain of the Semantic Web, where Datalog has been utilized for reasoning about RDF (Resource Description Framework), RDFS (RDF Schema), and OWL data [13].

Datalog's compatibility with large datasets and ability to express complex queries and recursive rules have contributed to its popularity. It has become the go-to choice for various applications, including database management systems, knowledge representation in artificial intelligence, and the Semantic Web. Its unique simplicity and expressiveness have allowed users to easily tackle complex problems, resulting in more efficient and effective solutions [14].

Based on first-order logic, Datalog employs predicates, variables, and constants to express rules and facts. A Datalog program consists of a finite set of rules and facts, where each rule is defined as a clause containing a head and a body. The head consists of a single positive literal, while the body is a conjunction of positive or negative literals [11].

In Datalog, uppercase letters represent variables, while lowercase letters or numbers signify constants. A fact is a ground atom devoid of variables, and a query is a conjunction of literals containing at least one variable. The semantics of a Datalog program is delineated by its minimal Herbrand model, representing the complete set of ground atoms that can be derived from the program's rules and facts [15].

The simplicity and expressiveness of Datalog's syntax make it a powerful tool for knowledge representation and querying in various domains. Rules and facts in Datalog can be easily defined and combined to create complex queries and recursive relationships [11]. This enables users to efficiently represent and manipulate large datasets, improving understanding and control over the data. Furthermore, the minimal Herbrand model provides clear and concise semantics for Datalog programs, allowing for efficient evaluation and optimization of queries [15].

One of its most significant advantages is the ability to express complex relationships and constraints using Datalog's syntax. It allows users to define complex queries, constraints, and views on relational data, enabling a higher level of understanding and control over the data. [15] This is particularly beneficial in the domain of the Semantic Web, where Datalog has been utilized for reasoning about RDF (Resource Description Framework) and RDFS (RDF Schema) data [16].

By leveraging Datalog's powerful capabilities, researchers and practitioners in the field of artificial intelligence have been able to model complex relationships and structures, thereby enhancing their understanding of the underlying data.

Datalog's syntax supports both positive and negative literals in the body of a rule, enabling users to express what is true and what is false or unknown. This feature is crucial for handling incomplete or noisy data, often encountered in real-world applications. [14], [16]

By providing a flexible and expressive syntax for knowledge representation, Datalog allows users to model various relationships, constraints, and uncertainties, making it a versatile and powerful tool for a wide range of applications [15] [16]. As a result, Datalog has emerged as an integral tool for processing and understanding semantic information within web-based resources. Datalog also finds applications in program analysis, encompassing static analysis, model checking, and compiler optimization [17]. This efficient reasoning allows for an improved understanding of program behavior and aids in developing more robust, optimized software.

In addition to the domains mentioned earlier, Datalog is employed in network management, access control, and configuration management to express policies, rules, and constraints concisely and declaratively [18]. Network management uses a declarative networking language called Network Datalog (NDlog) to define routing policies, security rules, and performance constraints, enabling efficient and effective management of complex network infrastructure. In access control, Datalog specifies and enforces security policies, ensuring that only authorized users have access to sensitive resources [18].

1.1.2. Inference

Inference is a fundamental concept in both formal logic and computational reasoning systems. It refers to the process of deriving new propositions logically implied by a given set of propositions [19]. Inference is guided by rules or axioms defining the logical relations between different propositions. These rules form the basis of a formal inference system, which can be used to systematically generate all logically valid inferences from a given set of input propositions [19], [20].

A classic example of inference comes from propositional logic, where an inference rule called *modus ponens* allows us to derive a proposition p from the propositions $p \to q$ and q.

This can be formalized as follows:

$$\frac{p \to q, q}{\therefore p}$$ [19].

Another example comes from first-order logic, where the universal instantiation rule allows us to derive a proposition $P(a)$ from the universal proposition $\forall x.P(x)$. This can be formalized as:

$$\frac{\forall x.P(x)}{\therefore P(a)}$$ [19].

Finally, in the context of RDF and RDFS, an inference rule might allow us to derive a triple *(a, rdf : type, y)* from the triples *(a, rdf:type, x)* and *(x, rdfs:subClassOf, y)*. This corresponds to the subclass inheritance rule in RDFS and can be formalized as:

$$\frac{(a, rdf : type, x), (x, rdfs : subClassOf, y)}{\therefore (a, rdf : type, y)}$$ [20].

These examples illustrate the principle of inference in different logical systems. Despite their differences, all these systems share the goal of deriving new information logically consistent with the given input information. In this way, inference is the foundation of logical reasoning and knowledge discovery from formal proofs and theorem proving to data analysis and machine learning. [20] [19]

1.1.3. Materialization

Materialization is a powerful reasoning technique that involves the computation and explicit storage of all possible inferences derived from a set of facts and rules. This method contrasts query-driven reasoning, where inferences are generated dynamically during query processing [11], [21].

Given a set of facts and rules represented as Datalog predicates and clauses:

1. Facts: *Parent(John, Jim), Parent(John, Ann).*
2. Rule: *Sibling(X, Y) ← Parent(Z, X), Parent(Z, Y) [21].*

Materialization would involve computing all instances of the Sibling predicate based on the Parent facts and storing them in the database, resulting in *Sibling(Jim, Ann)* and *Sibling(Ann, Jim).*

Materialization provides numerous benefits, including simplifying query processing by precomputing all pertinent inferences, resulting in faster query response times. However, materialization also necessitates considerable computational resources and storage capacity from the outset, as all inferred facts must be calculated and stored prior to the commencement of query processing [16].

The overhead associated with materialization can pose challenges in domains characterized by large datasets or frequent updates. Nevertheless, the advantages of materialization frequently surpass the drawbacks in situations where swift query response times are paramount [16]. By precomputing and storing all inferred facts, materialization reduces the complexity of query processing, enabling systems to access relevant facts without directly needing on-the-fly reasoning [17]. This characteristic streamlines the process and allows for more efficient use of computational resources, making it an attractive option for many applications.

The ability to fully exploit the range of inferences contributes to the data's overall value. It can potentially unlock new insights and opportunities that might remain hidden[17]. However, materialization also presents several challenges that must be considered. Another challenge associated with materialization is its suitability for highly dynamic datasets. In cases where updates are frequent, the constant recomputation and storage of inferred facts can become a resource-intensive process, potentially negating the benefits offered by materialization [17], [18]. In such scenarios, alternative reasoning techniques or hybrid approaches may be more suitable to balance the trade-offs between materialization and query-driven reasoning[18].

Despite these challenges, materialization remains an essential technique for various reasoning tasks, providing valuable benefits for various applications. By carefully considering each application's specific requirements and constraints, it is possible to make informed decisions about the appropriate use of materialization and its potential advantages [20].

Materialization enhances the expressiveness of queries over RDF and RDFS data and streamlines query processing by precomputing and storing inferences. This precomputation leads to faster query response times, as relevant inferred facts can be directly accessed without needing on-the-fly reasoning. Consequently, materialization helps improve the overall efficiency and effectiveness of RDF and RDFS-based systems, making them better equipped to handle the demands of large-scale data processing and reasoning tasks [17].

The process can be computationally intensive, especially for large datasets with complex schemas. Generating and storing all inferred triples may require significant processing power and storage capacity, which can be a constraint for systems with limited resources [16]. Additionally, materialization may not be well-suited for highly dynamic datasets, as frequent updates to the data could necessitate constant recomputation and storage of inferred facts, potentially negating the benefits offered by materialization [17]. Despite these challenges, materialization remains vital in RDF and RDFS reasoning.

In many cases, materialization can significantly improve the performance of RDF and RDFS-based systems, enabling more powerful and expressive querying of the underlying data [15] [16]. As the Semantic Web continues to evolve and grow, materialization will undoubtedly play a crucial role in realizing its full potential, supporting advanced reasoning capabilities over vast, interconnected datasets, and facilitating a more intelligent, data-driven web experience.

As the Semantic Web evolves, materialization will be critical in realizing its full potential. Materialization enables advanced reasoning capabilities over large, interconnected datasets by providing a more complete and consistent representation of RDF and RDFS data [16]. In turn, this facilitates more intelligent, data-driven web experiences and supports data integration, sharing, and querying across heterogeneous sources.

1.2. Semantic Technologies

1.2.1. RDF

1.2.1.1. Overview of RDF

The Resource Description Framework (RDF) is an essential technology that underpins the Semantic Web, offering a standard graph-based data model for representing and exchanging data on the Internet. As a core component of the Semantic Web, RDF plays a crucial role in enabling the encoding, integration, and sharing of structured and semi-structured data from various sources [22]. This ability to bring together data from various domains fosters interoperability, making it possible for disparate systems and applications to communicate and understand one another more effectively. In doing so, RDF paves the way for the creation of vast linked data networks, which can be leveraged to enhance information sharing and facilitate the discovery of new insights across the web [22].

RDF's graph-based data model represents data as a collection of subject-predicate-object statements, known as triples. Each triple in the RDF graph describes a specific relationship between two resources or a resource and a constant value, establishing a directed, labeled graph that captures the complex connections and dependencies between various pieces of information [22]. RDF relies on Uniform Resource Identifiers (URIs) to ensure uniqueness and universality in representing resources within the graph. At the same time, blank nodes are used in cases where a unique identifier is unnecessary. Constant values such as strings or numbers are represented using literals [22].

This flexible and extensible data model allows RDF to accommodate a broad range of data types and structures. It is well-suited for the Semantic Web's diverse and dynamic information landscape. By providing a common framework for representing data from multiple sources, RDF makes it possible to integrate and consolidate information in previously unattainable ways [23] [22]. This integration capability is precious in the age of Big Data, where the sheer volume, variety, and velocity of information pose significant challenges for traditional data management and processing techniques [13], [16], [23].

RDF's standardized data model also simplifies the process of exchanging data between systems and applications, eliminating many of the complexities associated with data conversion and translation [20]. By adhering to a universally recognized representation format, RDF enables seamless data exchange across different platforms, fostering greater collaboration and information sharing among various stakeholders on the web. This, in turn, contributes to the development of more robust and interconnected data networks that can be harnessed for a wide range of purposes, from scientific research and business intelligence to social networking and digital humanities [9]. Its graph-based data model, combined with its capacity to encode, integrate, and share data from diverse sources, has laid the foundation for the creation of extensive linked data networks that transcend traditional boundaries and enable novel forms of collaboration, discovery, and innovation [19], [22]. As the Semantic Web continues to evolve and expand, RDF's role as a unifying force for data representation and exchange will remain as critical as ever, paving the way for new opportunities and advancements in the digital realm.

1.2.1.2. RDF Data Model and Syntax

The Resource Description Framework (RDF) data model is built around triples, composed of subject-predicate-object statements. These triples serve as the building blocks for constructing a directed, labeled graph that represents relationships between various resources in a structured and easily understandable manner [20], [22]. In this graph, resources, which can be subjects or objects, are connected by properties, also known as predicates. This design represents complex connections and dependencies between different pieces of information, providing a powerful tool for encoding, integrating, and sharing structured and semi-structured data from various sources [22].
The core data model can be precisely delineated as follows [24]:

1. There is a set of *Nodes*, denoted as *N*.
2. A subset of *N*, referred to as *PropertyTypes*, is denoted as *P*.
3. A set of 3-tuples, named *T*, is present. These elements are informally regarded as properties. Each tuple's first item is an element of *P*, the second belongs to

N, and the third can be an element of *N* or an atomic value, such as a Unicode string [24].

Within this data model, the resources being characterized and the values describing them serve as nodes in a directed labeled graph. Interestingly, values can also be resources. The arcs connecting node pairs correspond to the property type names [24]. This can be visually represented.

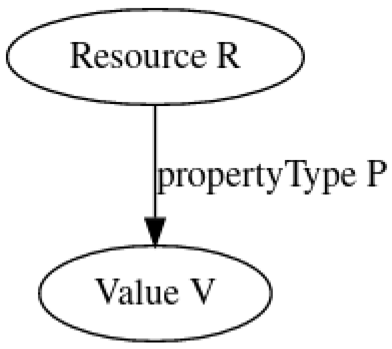


Figure 3. Relation of Resource R to Value V. [24]

This can be interpreted as "V is the value of the property P for resource R" or in a left-to-right fashion, "R has property P with value V."

To illustrate, consider the straightforward statement:

```
"Ora Lassila" is the "author" of the webpage
"http://www.w3.org/People/Lassila"[24]
```

This statement can be depicted in the following manner:

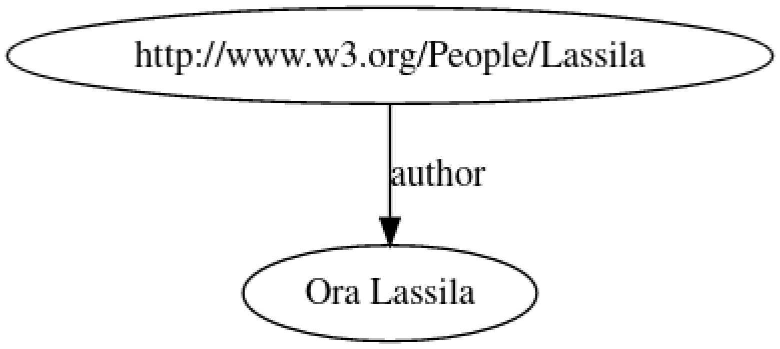


Figure 4. Relation of Ora Lassila as an author of http://www.w3.org/People/Lassila. [24]

Here, the notation [URI] signifies the instance of the resource identified by URI, and "..." indicates a simple Unicode string. [24] Following the formal definition, the property "author", i.e., the arc labeled "author" along with its source and target nodes, constitutes the triple (3-tuple):

```
{author, [http://www.w3.org/People/Lassila], "Ora Lassila"}
```

In this context, "author" represents a node used for labeling this arc. This model formulation is conducive to reification, implying that the relationship expressed by the arc can be transformed into a concrete node that can be referenced, as shown below [24]:

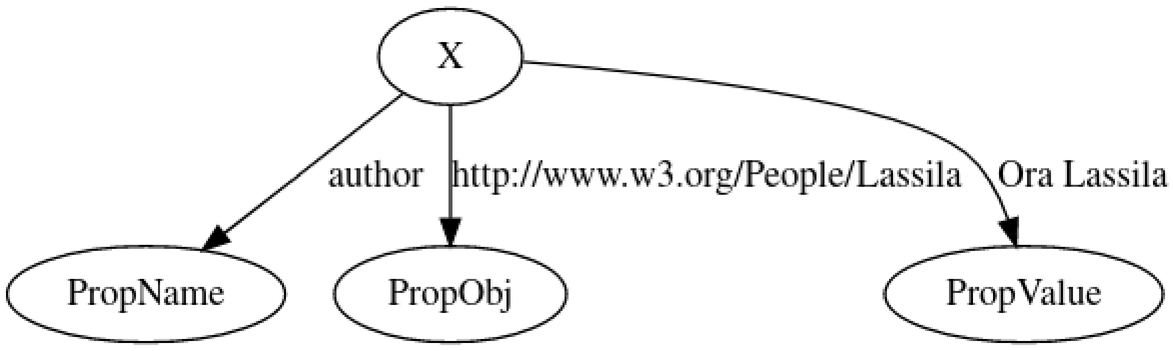


Figure 5. Relationship between the concrete node, its *PropName*, *PropObj,* and *PropValue.* [24]

This implies the addition of a node X and three new triples:

```
{PropName,  X, author}
{PropObj,   X, [http://www.w3.org/People/Lassila]}
{PropValue, X, "Ora Lassila"} [24]
```

It is subsequently demonstrated that reification enables the expression of modalities (such as beliefs about statements) or simply the attachment of any properties to other properties [24].

A set of triples sharing the same second item is an assertion. Assertions are particularly beneficial when enumerating several properties of the same resource. Assertions are diagrammed as follows:

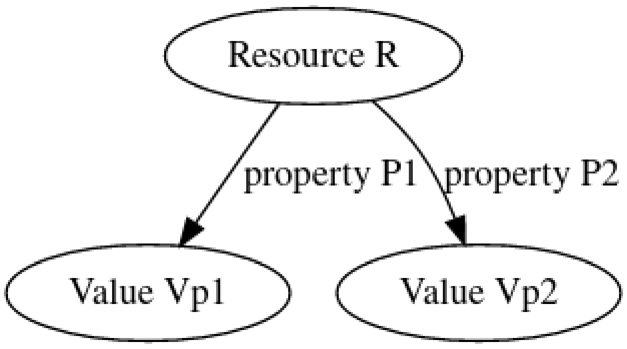


Figure 6. An assertion showing Resource *R* as a property of Values *Vp1* and *Vp2 [24]*.

An RDF assertion can be a resource and, therefore, be described by properties; in other words, an assertion can serve as the source node of an arc [24].

Assertions may be associated with the resource they describe in one of the following four ways [24]:

1. The assertion may be embedded within the resource.
2. The assertion may be external to the resource but provided by the transfer mechanism in the same retrieval transaction as the one returning the resource (referred to as "along-with").
3. The assertion may be retrieved independently from the resource, potentially from a different source (referred to as "service bureau").
4. The assertion may encapsulate the resource [24].

To ensure uniqueness and universality within the RDF graph, resources are typically represented using Uniform Resource Identifiers (URIs) [25]. URIs serve as globally unique identifiers that reference and distinguish resources across different systems and applications, promoting interoperability and facilitating data exchange across diverse platforms. In cases where a unique identifier is not necessary or applicable, RDF allows using blank nodes, which serve as placeholders for resources without explicitly identifying them [23] [22].

On the other hand, constant values such as strings, numbers, or dates are represented in RDF graphs using literals. Literals provide a standardized way of encoding these values, ensuring they can be consistently interpreted and processed by various systems and applications that work with RDF data. By adopting a uniform representation for constant values, RDF simplifies data integration and exchange, allowing for more efficient and effective communication between different components of the Semantic Web [23].

With its subject-predicate-object triple-based design, the RDF graph's structure provides a high degree of flexibility and extensibility. This versatile data model can accommodate various data types and structures, making it particularly well-suited for the diverse and dynamic landscape of the Semantic Web [22]. RDF enables the integration and consolidation of information in previously unattainable ways by offering a common framework for representing data from multiple sources. This capability is precious in the age of Big Data, where traditional data management and processing techniques struggle to cope with the vast volume, variety, and velocity of information being generated and exchanged [22].

1.2.1.3. RDF Serialization Formats

Resource Description Framework (RDF) data can be serialized into various formats to cater to different requirements and preferences [22]. These formats, including RDF/XML, Turtle, N-Triples, and JSON-LD, provide distinct benefits regarding readability, compactness, and compatibility with other web technologies. Each serialization format addresses specific use cases and caters to the diverse needs of developers, data practitioners, and researchers working with RDF data [23].

RDF/XML is one of the earliest and most widely supported serialization formats for RDF data [26]. It leverages the ubiquity and versatility of XML, a markup language widely used for data exchange on the web. While RDF/XML is powerful and comprehensive, its verbose nature and complex syntax can make it harder to read and understand, especially for human users[26]. This verbosity can also increase data size, impacting performance when working with large RDF datasets or transmitting data over networks[26].

Example 1. A RDF triple in XML format.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ex="http://example.com/stuff/1.0/">
<rdf:Description
rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
  <ex:editor>
    <rdf:Description ex:homePage
rdf:resource="http://purl.org/net/dajobe/"/>
  </ex:editor>
</rdf:Description>
</rdf:RDF>
```

Turtle (Terse RDF Triple Language) is an alternative serialization format that offers a more human-readable and compact representation of RDF data[27]. Designed as a more concise counterpart to RDF/XML, Turtle uses a simplified, text-based syntax that is easier to read and write. By eliminating the verbosity and complexity associated with RDF/XML, Turtle enables users to work with RDF data more efficiently and effectively, making it a popular choice for developers and data practitioners seeking a more user-friendly option [27].

Example 2. A RDF triple in Turtle format.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.com/stuff/1.0/> .
<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  ex:editor [
    ex:homePage <http://purl.org/net/dajobe/>
  ] .
```

N-Triples is another RDF serialization format emphasizing simplicity and ease of use [28]. Like Turtle, N-Triples is a text-based format that provides an unambiguous representation of RDF triples. However, N-Triples is even more minimalistic than Turtle, offering a line-based, plain text format well-suited for processing large datasets and working with RDF data in automated environments. N-Triples are often used for tasks that require robustness and reliability, such as data validation, testing, or data exchange between systems [28].

Example 3. A RDF triple in N-triple format.

```
<http://www.w3.org/TR/rdf-syntax-grammar>
<http://purl.org/dc/elements/1.1/title>
"RDF/XML Syntax Specification (Revised)" .

<http://www.w3.org/TR/rdf-syntax-grammar>
<http://example.com/stuff/1.0/editor>
_:bnode .

_:bnode
<http://example.com/stuff/1.0/homePage>
<http://purl.org/net/dajobe/> .
```

JSON-LD (JSON for Linked Data) is a more recent RDF serialization format that builds upon the widely used JSON data interchange format. By combining the simplicity and accessibility of JSON with the expressiveness of RDF, JSON-LD offers a highly compatible and easily consumable representation of RDF data [29]. JSON-LD is particularly well-suited for web developers and applications that leverage JavaScript and other web technologies, making it an attractive option for those looking to incorporate RDF data into modern web-based environments [29].

Example 4. A RDF triple in JSON-LD format.

```
{
  "@context": {
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "dc": "http://purl.org/dc/elements/1.1/",
    "ex": "http://example.com/stuff/1.0/"
  },
  "@id": "http://www.w3.org/TR/rdf-syntax-grammar",
  "dc:title": "RDF/XML Syntax Specification (Revised)",
  "ex:editor": {
    "ex:homePage": {"@id": "http://purl.org/net/dajobe/"}
  } }
```

1.2.1.4. Querying RDF Data with SPARQL

SPARQL, the SPARQL Protocol, and RDF Query Language is a universal and standardized query language designed for RDF data. It enables users to retrieve, manipulate, and transform information stored in RDF graphs, unlocking the full potential of RDF data and facilitating more powerful, expressive querying capabilities [30] [31]. SPARQL's extensive features and support for various query patterns make it an indispensable tool for data practitioners, researchers, and developers working with RDF and the Semantic Web

One of the key strengths of SPARQL is its ability to support a wide range of query patterns. Basic graph pattern matching lies at the core of SPARQL queries, allowing users to specify patterns of interest in the form of subject-predicate-object triples[30]. By matching these patterns against the RDF graph, SPARQL can efficiently retrieve and return the relevant information in a structured format. This feature empowers users to ask sophisticated questions about the data and uncover insights that may not be immediately apparent through simpler query mechanisms[30].

Example 5. A simple SPARQL query retrieves all triples from an RDF graph [30].

```
SELECT ?subject ?predicate ?object
WHERE {
  ?subject ?predicate ?object .
}
```

This query uses variables (*?subject, ?predicate, and ?object*) to represent each triple's subject, predicate, and object in the graph. The *WHERE* clause specifies the pattern to match against the graph, and the *SELECT* clause indicates which variables' values to return in the query results[30].

In addition to basic graph pattern matching, SPARQL supports more advanced query constructs, such as optional and alternative patterns. Optional patterns enable users to request additional information that may or may not be present in the RDF graph without affecting the overall query results. This flexibility allows for more comprehensive querying and the extraction of richer information from the data[30]. Alternatively, alternative patterns allow users to specify multiple patterns that could satisfy the query, increasing the chances of retrieving relevant information even when some of the requested data is missing or incomplete[30].

Another powerful feature of SPARQL is its support for complex filters and aggregations[31]. Filters enable users to refine query results based on specific criteria, such as value ranges, string patterns, or mathematical expressions. By applying filters to query results, users can hone in on the most relevant and meaningful data, eliminating noise and improving the overall quality of the results[31]. Conversely, aggregations allow users to group and summarize data, computing metrics such as counts, averages, or sums. Aggregations are particularly useful for analyzing large datasets and identifying trends, patterns, or relationships within the data[31].

Example 6. A SPARQL query with a filter condition.

```
SELECT ?name ?email
WHERE {
  ?person a foaf:Person .
  ?person foaf:name ?name .
  OPTIONAL { ?person foaf:mbox ?email . }
  FILTER (!BOUND(?email))
}
```

In this query, the *OPTIONAL* clause is used to specify an optional pattern that matches a person's email address. The *FILTER* clause is then used to filter out any results where the email address is absent (i.e., the *?email* variable is not bound to a value)[31].

Federated querying is a unique capability of SPARQL that sets it apart from many other query languages. It enables users to query data from multiple RDF sources simultaneously, seamlessly integrating information from disparate sources and facilitating more comprehensive querying and data analysis [32].

Example 7. A SPARQL query that uses federated querying to retrieve data from multiple sources.

```
SELECT ?name ?birthPlace
WHERE {
  ?person dbpedia-owl:birthPlace ?birthPlace .
  SERVICE <http://example.org/sparql> {
    ?person foaf:name ?name .
  }
}
```

In this query, the *SERVICE* clause is used to specify a remote SPARQL endpoint from which to retrieve additional data. This allows the query to combine data from the local RDF graph (i.e., the person's birthplace) with data from the remote endpoint (i.e., the person's name). [32] This feature is especially important in the context of the Semantic Web, where data is often distributed across numerous interconnected repositories. By allowing users to access and combine data from multiple sources, federated querying helps overcome the limitations imposed by data silos. It promotes a more holistic approach to data analysis and knowledge discovery [32].

1.2.2. RDFS

1.2.2.1. Introduction to RDFS

RDF Schema (RDFS) is a lightweight ontology language designed to extend the Resource Description Framework (RDF), providing a standardized set of vocabulary and semantics for describing classes, properties, and constraints on RDF data [33], [34]. As a cornerstone for more expressive ontology languages like the Web Ontology Language (OWL), RDFS offers foundational reasoning capabilities over RDF graphs, enriching the expressiveness of RDF data and paving the way for more advanced knowledge representation and reasoning on the Semantic Web [33].

The primary goal of RDFS is to supply a means of defining the structure and semantics of RDF data, thereby enhancing its interpretability and facilitating more meaningful querying and data manipulation[33]. RDFS achieves this by introducing a basic vocabulary and a set of modeling constructs for defining classes, properties, and their relationships. This enables users to create hierarchies of classes and properties, define domain and range constraints, and specify additional metadata about resources in the RDF graph[33].

Classes in RDFS represent sets of resources that share common characteristics, and they can be organized hierarchically using the *rdfs:subClassOf* property. This hierarchical structure allows for the inheritance of properties and constraints from parent classes to child classes, promoting consistency and reusability of the schema. Properties in RDFS, on the other hand, are used to describe relationships between resources and can also be organized hierarchically using the *rdfs:subPropertyOf* property. This enables inheriting characteristics and constraints from parent to child properties[33]–[35].

Example 8. Defining a class "Person" in RDFS.

```
<rdfs:Class rdf:ID="Person" />
```

This line of RDF code defines a new class named "Person" using the *rdfs:Class* construct. The ability to define domain and range constraints for properties is another important feature of RDFS. Domain constraints specify the class of resources a property can use, while range constraints define the class of resources or literals that can target a property. These constraints help ensure the consistency of RDF data by enforcing the proper use of properties and relationships, and they provide valuable information for reasoning and querying processes [34].

Example 9. Using rdfs:subClassOf for creating class hierarchies.

```
<rdfs:Class rdf:ID="Employee">
  <rdfs:subClassOf rdf:resource="#Person" />
</rdfs:Class>
```

This code specifies that "Employee" is a subclass of "Person", meaning that every "Employee" is also a "Person".

Example 10. Defining properties and their constraints.

```
<rdf:Property rdf:ID="hasName">
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range
rdf:datatype="http://www.w3.org/2001/XMLSchema#string" />
</rdf:Property>
```

Here, *rdfs:domain* specifies that the "hasName" property can be used with instances of the "Person" class. In contrast, *rdfs:range* specifies that the values of this property should be strings.

In addition to its modeling constructs, RDFS introduces a set of built-in classes and properties for describing metadata and additional semantics about resources, classes, and properties. For example, *rdfs:label* and *rdfs:comment* can be used to attach human-readable labels and descriptions to resources, while *rdfs:seeAlso* and *rdfs:isDefinedBy* can be employed to link resources to additional documentation or definitions. These built-in vocabulary elements enrich the expressiveness of RDF data and facilitate more effective communication and knowledge sharing among data consumers [34].

Example 11. Defining properties and their constraints.

```
<rdfs:Class rdf:ID="Person">
  <rdfs:label>Person</rdfs:label>
  <rdfs:comment>A class representing persons.</rdfs:comment>
</rdfs:Class>
```

By enabling users to define classes, properties, and constraints on RDF data, RDFS enhances the interpretability and usefulness of RDF graphs, laying the groundwork for more sophisticated knowledge representation, querying, and reasoning on the Semantic Web [35] [33].

1.2.2.2. RDFS Vocabulary and Semantics

RDFS, or RDF Schema, is a lightweight ontology language that provides a predefined vocabulary for defining classes, properties, and their relationships within an RDF graph[34]. The primary goal of RDFS is to facilitate the structuring and organization of RDF data, making it more interpretable, meaningful, and easier to query. The RDFS vocabulary includes a set of foundational terms, such as *rdfs:Class*, *rdfs:subClassOf*, *rdf:Property*, *rdfs:domain*, and *rdfs:range*, that form the building blocks for creating hierarchies of classes and properties, as well as defining domain and range constraints [34].

The *rdfs:Class* term defines a class or a set of resources with common characteristics. Classes can be organized hierarchically using the *rdfs:subClassOf* property, which allows child classes to inherit properties and constraints from their parent classes. This inheritance mechanism promotes reusability and consistency throughout the schema[34]. The *rdf:Property* term defines properties that describe relationships between resources in an RDF graph. Like classes, properties can also be organized hierarchically using the *rdfs:subPropertyOf* property. This property enables child properties to inherit characteristics and constraints from their parent properties, ensuring a coherent structure within the schema[34].

Domain and range constraints play a vital role in RDFS, as they help to enforce the proper use of properties and relationships within an RDF graph. The rdfs:domain property is used to specify the class of resources with which a particular property can be associated. In contrast, the *rdfs:range* property defines the class of resources or literals that can be the target of a property[34]. These constraints contribute to the consistency and coherence of RDF data by providing guidelines for property usage and facilitating more precise querying and reasoning[34].

RDFS semantics are built on inference rules that describe the logical implications of using the RDFS vocabulary within an RDF graph[34]. These rules outline how new facts can be derived from existing data based on the relationships and constraints defined by the RDFS vocabulary. For instance, if a resource is an instance of class *A* and class *A* is a subclass of class *B*, the resource can also be inferred to be an instance of class B. Similarly, suppose property *P1* is a sub-property of property *P2*, and a resource *R1* has a relationship with resource *R2* using property *P1*. In that case, it can be inferred that *R1* also has a relationship with *R2* using property *P2[34]*.

Example 12. Using RDFS inference rules.

Given the following data:

```
<rdf:Description rdf:about="#HarryPotter">
  <hasAuthor rdf:resource="#JKRowling" />
</rdf:Description>

<rdfs:Class rdf:ID="FictionBook">
  <rdfs:subClassOf rdf:resource="#Book" />
</rdfs:Class>

<rdf:Description rdf:about="#HarryPotter">
  <rdf:type rdf:resource="#FictionBook" />
</rdf:Description>
```

If it is established that "FictionBook" is a subclass of "Book," it can be logically inferred that "HarryPotter", being an instance of "FictionBook," is also an instance of "Book." Such deductive rules empower RDFS reasoners to conduct elementary inference and materialization operations, thereby amplifying the expressiveness and applicability of RDF data [34].

1.2.2.3. RDFS Inference and Reasoning

RDFS inference is a fundamental process in the Semantic Web that deals with deriving new triples from an RDF graph by utilizing the RDFS schema and its underlying semantics[34]. This process is essential for harnessing the full potential of RDF and RDFS as it enables the discovery of implicit knowledge and relationships within the data, enriching the graph and enhancing its consistency and expressiveness. RDFS reasoning is the core mechanism of this inference, as it employs a set of well-defined inference rules based on the RDFS vocabulary and semantics[34].

Subclass and property inheritance are two prominent RDFS inference rules crucial in reasoning. Subclass inheritance pertains to the hierarchical organization of classes in RDFS, where a child class is considered a subset of its parent class[34]. The RDFS denotes this relationship:subClassOf property. When an RDF graph contains an instance of a particular class, RDFS reasoning can infer that the instance also belongs to any superclass of that class. This inference allows for more comprehensive and accurate querying, as it captures the implicit relationships between instances and classes in the RDF graph[34].

On the other hand, property inheritance deals with the hierarchical organization of properties in RDFS. Like classes, properties can be organized into a hierarchy using the rdfs:subPropertyOf property[34]. Property inheritance enables properties to inherit characteristics and constraints from their parent properties, which helps maintain a consistent and coherent structure within the RDF graph[34]. When an RDF graph contains a triple with a specific property, RDFS reasoning can infer additional triples with the same subject and object but using a super property of the original property. This inference enriches the RDF graph by identifying implicit relationships between resources based on their property hierarchy[34].

In addition to subclass and property inheritance, RDFS reasoning also considers domain and range constraints associated with properties. These constraints, denoted by rdfs:domain and rdfs:range properties, help enforce the proper use of properties and relationships within an RDF graph. RDFS reasoning can derive new facts based on these constraints, further enhancing the completeness and consistency of the RDF graph[34].

1.2.2.4. RDFS as a Set of Datalog Rules

Integrating RDFS with Datalog provides a powerful and efficient method for reasoning over RDF data. By representing RDFS as a set of Datalog rules, the RDFS vocabulary and semantics can be effectively translated into Datalog predicates and clauses [17]. This representation allows for efficient RDFS reasoning using Datalog-based engines and facilitates seamless integration with other Datalog-based systems and applications, expanding the scope and utility of both RDFS and Datalog in the context of the Semantic Web [36].

As a declarative logic programming language, Datalog is well-suited for expressing complex queries and recursive rules over large datasets. By transforming RDFS into Datalog rules, the inference capabilities of RDFS can be harnessed by Datalog-based systems to reason about RDF data efficiently. This transformation bridges the gap between the two paradigms, allowing for a unified approach to knowledge representation and reasoning that combines both technologies' strengths [36].

Example 13. Translating RDFS axioms into Datalog rules:

```
rdfs2(X, Y) :- rdf(X, rdf:type, Y).
rdfs3(X, Y) :- rdf(X, rdfs:subClassOf, Y).
rdfs5(X, Y) :- rdf(X, rdfs:subPropertyOf, Y).
rdfs6(X, Y) :- rdf(X, rdfs:domain, Y).
rdfs7(X, Y) :- rdf(X, rdfs:range, Y).

rdfs9(X, Y) :- rdfs2(X, Z), rdfs3(Z, Y).
rdfs10(X, Y) :- rdf(X, Z, Y), rdfs5(Z, W).
rdfs11(X, Y) :- rdf(X, Z, _), rdfs6(Z, Y).
rdfs12(X, Y) :- rdf(_, Z, X), rdfs7(Z, Y).
```

In this example, the RDFS axioms are represented as Datalog rules where *X*, *Y*, and *Z* are variables, *rdf(X, rdf:type, Y)*, *rdf(X, rdfs:subClassOf, Y)*, etc., are RDF triples and *rdfs2*, *rdfs3*, etc., are RDFS inferences. One significant advantage of representing RDFS as Datalog rules are the optimization potential Datalog engines offer. These engines employ various optimization techniques to process complex queries and rules efficiently. By leveraging these optimizations, RDFS reasoning over large RDF graphs can be performed more effectively, addressing some scalability challenges associated with RDF and RDFS technologies. [17]

1.3. Link Prediction

1.3.1. Definition and Importance

Link prediction (LP), a crucial problem in graph analytics and network science, focuses on determining the probability of connections between nodes in a graph by analyzing its structure and properties. [37] This process involves predicting the missing entity in a relational triple, either in the form of ⟨*h, r, ?*⟩ (known as tail prediction) or ⟨*?, r, t*⟩ (known as head prediction). To simplify, the known entity in the prediction is referred to as the source entity, while the entity to be predicted is termed the target entity. [37]

Over time, many strategies have been proposed to solve the LP task. Some of these methods are based on observable features and use techniques like Rule Mining or the Path Ranking Algorithm to identify missing relationships (or triples) in the graph [37]. Recently, as Machine Learning techniques have advanced, there has been an interest in discovering latent features in the graph by creating vectorized representations, also known as embeddings, of the graph's components. In broad terms, embeddings are vectors containing numerical values that can represent different types of elements (for example, words, individuals, products, and so on, depending on the specific domain) [2], [37].

These embeddings are automatically learned based on how the respective elements appear and interact with each other in representative real-world datasets. In social network analysis, link prediction predicts friendships, interactions, or other relationships between individuals in a network. By analyzing the network structure and properties, link prediction can determine the likelihood of future connections or uncover existing connections that may have been previously unknown [2]. This can be particularly useful in understanding the dynamics of online social networks, facilitating targeted marketing campaigns, or aiding in community detection and analysis [37].

Recommender systems are another area where link prediction has significant applications. These systems aim to provide users with personalized products, content, or service recommendations based on their preferences and behavior. Link prediction can help identify potential connections between users and items, suggesting more likely relevant and interesting recommendations. In e-commerce platforms, for example, link prediction can recommend products likely to be purchased together or suggest content that users with similar preferences have enjoyed [38].

1.3.2. Traditional link prediction Techniques

Statistical methods for link prediction constitute a widely used approach to infer potential connections between nodes in a graph [39] [40]. These methods primarily focus on determining node similarity scores by examining their properties, attributes, or neighborhood structures. The underlying assumption is that a higher similarity between nodes increases the likelihood of an edge existing between them [40].

One commonly employed statistical measure for link prediction is the Jaccard coefficient [41]. This metric calculates the similarity between two nodes by considering the ratio of their shared neighbors to the total number of distinct neighbors. In other words, it measures the overlap between the sets of neighbors for two nodes, giving a score between 0 and 1. A higher Jaccard coefficient indicates a higher degree of similarity between the nodes, suggesting a greater likelihood of a connection [41].

Cosine similarity is another widely used measure for link prediction in graph analytics. This approach computes the similarity between two nodes by measuring the cosine of the angle between their attribute vectors. The resulting score ranges from -1 to 1, with 1 representing perfect similarity, 0 indicating no similarity, and -1 suggesting a completely different relationship. Cosine similarity is beneficial when dealing with high-dimensional data, as it is less sensitive to the magnitude of the attribute vectors and focuses more on the directional relationship between them [42].

The Pearson correlation coefficient is an additional statistical method utilized for link prediction. This measure assesses the linear relationship between the attributes of two nodes by calculating the correlation between their attribute vectors. Pearson correlation coefficient values range from -1 to 1, where 1 indicates a strong positive correlation, -1 implies a strong negative correlation, and 0 suggests no correlation between the nodes' attributes. A high positive correlation may indicate a higher likelihood of a connection between the nodes, while a high negative correlation might suggest a lower probability of an edge [43].

1.3.3. Machine Learning-Based link prediction

In supervised machine learning techniques for link prediction, models are trained using labeled examples of node pairs and information about whether a connection exists between them. Based on these examples, the models learn to predict the presence or absence of connections, which can then be generalized to make predictions on previously unseen node pairs [44].

In the research study "Link Prediction Based on Graph Neural Networks" by Zhang and Chen (2018), supervised learning was used to address link prediction in citation networks[44]. The researchers employed Graph Convolutional Networks (GCNs) to capture local and global graph structures and node features. The model was trained on labeled pairs of nodes, and the results demonstrated strong predictive performance, outperforming several traditional methods[44]. Supervised techniques have demonstrated impressive accuracy and predictive performance in various link prediction tasks, often surpassing the capabilities of traditional statistical and graph-based methods[44].

On the other hand, unsupervised machine learning techniques focus on discovering patterns and structures within the graph without relying on explicit labels. Instead, they seek to learn meaningful representations of nodes and their relationships that can be used to estimate the likelihood of connections [45]. Ou, Jin, and Yang (2016) used an unsupervised machine learning approach in the study "Asymmetric Transitivity Preserving Graph Embedding." They leveraged a graph embedding technique to learn latent representations of nodes in an unsupervised manner [45]. These representations were then used to predict missing links in the graph. The method performed superior over several baselines in various link prediction tasks, especially when explicit labels were not readily available[45].

In the research paper "Semi-Supervised Classification with Graph Convolutional Networks" by Kipf and Welling (2016), a semi-supervised learning approach was used for link prediction where labeled data was limited. The authors utilized Graph Convolutional Networks (GCNs) that combined the strengths of both supervised and unsupervised learning methods. The model was trained on a small set of labeled node pairs; the rest were unlabeled[46]. The results showed that the model could effectively generalize from the limited labeled data to predict the unlabeled node pairs.

These techniques often involve clustering or dimensionality reduction methods that can uncover latent structures and communities within the graph. Unsupervised methods have also succeeded in link prediction tasks, especially in cases where labeled training data is scarce or unavailable[46].

1.3.4. Graph Neural Networks

GNNs have risen as a potent class of machine learning models expressly crafted for graph-structured data. GNNs decipher intricate patterns and relationships between nodes by fusing local and global information from the graph [47]. They employ iterative message-passing mechanisms to disseminate information among neighboring nodes and update their embeddings or features, ultimately creating node representations that encapsulate local and global graph structures [47]. GNNs have showcased outstanding performance in various link prediction tasks, often surpassing traditional techniques and machine learning methods [47]. Their ability to learn comprehensive and expressive representations of nodes and their relationships makes them especially well-suited for link prediction challenges. In these problems, the graph's underlying structure is pivotal in determining the probability of connections between nodes.

While machine learning-based link prediction techniques have shown great promise, they also come with challenges and opportunities. One key challenge is the scalability of these methods, particularly for large-scale graphs with millions or billions of nodes and edges [46] [48]. Developing efficient and scalable algorithms that can handle such massive graphs remains an active area of research.

Additionally, the interpretability of machine learning models, intense learning models like GNNs, is another challenge that needs to be addressed. Understanding the rationale behind the predictions made by these models is crucial for building trust and ensuring their applicability in real-world scenarios for understanding complex networks and their underlying structures [47].

1.4. Summary

Reasoning, Semantic Technologies, and link prediction are critical components in the field of knowledge representation and the Semantic Web [33], [34], [48]. The Resource Description Framework (RDF) and RDF Schema (RDFS) are fundamental technologies that enable the encoding, integration, and sharing of structured and semi-structured data from diverse sources. They facilitate interoperability and the creation of linked data networks, enabling advanced reasoning capabilities over large, interconnected datasets [44] [43]. One essential technique for unlocking the full potential of linked data is materialization, generating and storing all possible inferences from a given RDF graph based on the RDFS schema [36].

Link prediction, a crucial problem in graph analytics and network science, involves inferring the likelihood of connections between nodes based on the graph's structure and properties. It has many applications, including social network analysis, recommender systems, bioinformatics, and fraud detection. Traditional link prediction techniques, such as statistical methods and graph-based approaches, exploit properties, attributes, or neighborhood structures of nodes, as well as topological properties of the graph, to estimate the likelihood of connections between nodes [48] [36], [44]. However, RDF data's rapidly growing scale and complexity necessitate more advanced techniques for reasoning and link prediction [2].

Deep learning-based methods, particularly Graph Neural Networks (GNNs), have emerged as powerful tools for addressing these challenges. GNNs are machine learning models specifically designed for graph-structured data [48] [46]. They can learn complex patterns and relationships between nodes by combining local and global information from the graph. GNNs use iterative message-passing mechanisms to propagate information between neighboring nodes, updating their embeddings or features and generating node representations that capture local and global graph structures [47].

Applying GNNs to RDFS materialization, a deep learning-based approach, can significantly improve the efficiency and effectiveness of reasoning and link prediction tasks. GNNs have demonstrated excellent performance in various link prediction tasks, often outperforming traditional methods and machine learning techniques. Their ability to learn rich and expressive representations of nodes and their relationships makes them particularly well-suited for tasks where the underlying graph structure is crucial in determining the likelihood of connections between nodes [44].

A deep learning-based RDFS materialization approach can potentially address several challenges associated with traditional methods, such as increased computational and storage requirements, handling noisy or incomplete data, and scalability[2]. By leveraging the power of GNNs, it is possible to develop more robust and efficient techniques for RDFS materialization that can handle large, complex, and dynamic datasets[2]. Furthermore, the use of supervised and unsupervised learning techniques in conjunction with GNNs offers promising avenues for improving the accuracy, completeness, and consistency of materialized RDF graphs [2] [47] [48].

2. Materialization with Deep Learning

2.1. Deep learning for noise-tolerant RDFS reasoning

The primary motivation for noise-tolerant RDFS reasoning is that the Semantic Web (SW) and the Web of data are inherently noisy. Traditional SW reasoning focuses on soundness and completeness, assuming that the input data is accurate [24], [49] [36]. However, this assumption does not hold in real-world scenarios, where data is often incomplete, inconsistent, or contradictory [2], [7]. Noise-tolerant reasoning is essential to address these challenges and enable more accurate and reliable RDFS materialization[2].

Bassem Makni and James Hendler's paper, "Deep learning for noise-tolerant RDFS reasoning," presents a novel approach that extends noise-tolerance in the SW to full RDFS reasoning[2]. The proposed method adapts Knowledge Graph (KG) embedding techniques to RDFS reasoning by layering RDF graphs and encoding them as 3D adjacency matrices. These matrices form "graph words" and sequences representing the input graph and its entailments. RDFS inference is then formulated as a translation of these graph word sequences, achieved through neural machine translation. This deep learning-based approach demonstrates noise tolerance that is not available with traditional rule-based reasoners [2], [9], [11].

The primary goal of the noise-tolerant RDFS reasoning approach is to enable full RDFS reasoning capable of handling noisy and incomplete data. By addressing these challenges, the approach provides more accurate and reliable materialization results, making it a promising alternative to traditional reasoning methods that assume perfect input data[2], [9]. To achieve this, the RDF graphs are organized into layers, each representing a specific aspect or property of the graph[2].

This layered structure is essential for efficiently encoding the graph data as 3D adjacency matrices, which serve as a compact graph representation and facilitate further processing[2]. The layered RDF graphs are then represented as 3D adjacency matrices, with each layer forming a "graph word." This compact representation reduces the storage requirements for the RDF graph data and allows for efficient manipulation and processing, paving the way for advanced reasoning techniques. Graph words and sequences are crucial in this approach[2].

Each input graph and its entailments are represented as sequences of graph words, which serve as the input and output for the neural machine translation process[2]. This representation enables a more straightforward and efficient translation between different graph structures and their corresponding entailments.

This approach employs neural machine translation, a deep learning technique, for RDFS inference [2]. By translating the sequences of graph words, neural machine translation effectively captures the relationships and entailments in the RDF graph, resulting in more accurate and noise-tolerant RDFS reasoning[2].

2.2. Comparison with logic-based and traditional methods

2.2.1. Limitations of Traditional RDFS Reasoners

Traditional RDFS reasoners have been designed to focus on soundness and completeness, assuming that the input data is accurate and without noise. However, this assumption often does not hold in real-world scenarios, where data can be incomplete, inconsistent, or contradictory [10]. These reasoners struggle to handle such noisy data, leading to inaccurate or incomplete materialization results. The challenge lies in developing reasoning methods that can effectively cope with the inherent noise in web data without sacrificing soundness and completeness [2], [9].

The Web of data is inherently noisy due to various factors, such as data entry errors, inconsistencies in data sources, and the dynamic nature of web data. This noise poses significant challenges for traditional RDFS reasoning methods designed to operate using accurate input data [2], [3]. The presence of noise can lead to incorrect inferences, impacting the overall quality and reliability of the materialization process. Addressing this challenge requires developing noise-tolerant reasoning techniques that can handle incomplete and inconsistent data while maintaining the desired level of soundness and completeness [2].

2.2.2. Adapting KG Embedding Techniques for RDFS Reasoning

Knowledge Graph (KG) embedding techniques have been widely used for link prediction but have not been directly applied to RDFS reasoning[2]. The problem setting in RDFS reasoning differs from link prediction, making it challenging to adapt these techniques for RDFS materialization. A key challenge is to develop a tailored approach that can effectively represent RDF graph structures and their entailments while capturing the specific constraints and relationships defined in the RDFS schema[2]. This requires a thorough understanding of the differences between link prediction and RDFS reasoning, as well as the development of novel embedding techniques and deep learning models that can handle the unique challenges of RDFS materialization [2], [6], [7].

2.2.3. Scalability and Performance Issues

Scalability and performance are critical concerns in RDFS reasoning, especially when dealing with large and complex RDF graphs [2]. Traditional rule-based reasoners can struggle with performance bottlenecks and high memory requirements, making them unsuitable for large-scale materialization tasks. Adapting deep learning techniques for RDFS reasoning introduces additional challenges, such as the computational complexity of neural network models and the need for specialized hardware resources like GPUs[2]. Addressing these scalability and performance issues requires the development of efficient algorithms, data structures, and deep-learning architectures that can handle the demands of large-scale RDFS materialization while maintaining acceptable noise-tolerance and reasoning accuracy.

2.2.4. Soundness and Completeness of Reasoners

Deterministic solutions, such as rule-based reasoners, primarily focus on soundness and completeness. Soundness ensures that any inferences drawn from the input data are valid, while completeness guarantees that all possible valid inferences can be derived. While these properties are essential for RDFS reasoning[2], [11], deterministic solutions often assume that input data is accurate and noise-free. This assumption may not hold in real-world scenarios, leading to potential inaccuracies and incomplete materialization results. Comparatively, noise-tolerant approaches like deep learning-based methods can better handle the inherent noise in web data while still aiming to maintain soundness and completeness[2].

2.2.5. Noise-Tolerant Reasoning in Type Inference vs. Full RDFS Reasoning

Noise-tolerant reasoning in type inference focuses on inferring the types of resources in RDF graphs despite noise [2]. While this is an essential step in RDFS reasoning, it does not cover the full scope of RDFS materialization, which involves reasoning over classes, properties, and their relationships [11]. In contrast, full RDFS reasoning, as explored in the deep learning-based approach, extends noise tolerance to handle type inference and other aspects of RDFS materialization[2]. This broader scope enables a more comprehensive understanding of the RDF graph structure and its entailments, providing a more effective solution to address noisy web data[2].

Rule-based reasoners are a popular deterministic solution for RDFS materialization[2], [9]. They rely on a predefined set of rules to infer new information from the RDF graph. While these reasoners are typically sound and complete, they may struggle to effectively handle noisy or incomplete data. Additionally, rule-based reasoners can suffer from scalability and performance issues, especially when processing large and complex RDF graphs. These limitations make it challenging for rule-based reasoners to handle real-world web data's dynamic and noisy nature [2], [5].

2.2.6. Advantages and Disadvantages of Deep Learning-Based Approach

The deep learning-based approach to RDFS materialization offers several advantages over deterministic solutions, such as handling noisy and incomplete data more effectively[2]. By leveraging advanced techniques like neural machine translation and graph embeddings, deep learning-based methods can capture complex relationships and constraints in RDF graphs, leading to more accurate and reliable materialization results [2]. Additionally, given the right hardware resources and optimized algorithms, deep learning models can potentially scale better than rule-based reasoners [2].

The computational complexity of deep learning models can be a significant challenge, as they often require specialized hardware and significant computational resources. Moreover, deep learning models may be more difficult to interpret and understand than rule-based reasoners, making it harder to diagnose errors or explain the reasoning process [2].

2.3. Visual Representation of the Approach

2.3.1. Iteration Algorithm

This algorithm is the main driving force of the entire RDF graph processing procedure. The algorithm iterates over all triples (subject, predicate, object) in the RDF graph, sorted by the property. For each triple, it checks if the predicate is in the properties dictionary. If it is not, the algorithm skips to the next triple. If the predicate is in the properties dictionary but not the functional properties dictionary, it is added to the functional properties dictionary. Then, the algorithm proceeds to add the subject and object resources related to the predicate, look up the IDs of the subject and object, and append these to the sparse encoding. The algorithm continues this iterative process until it has processed all triples in the RDF graph[2].

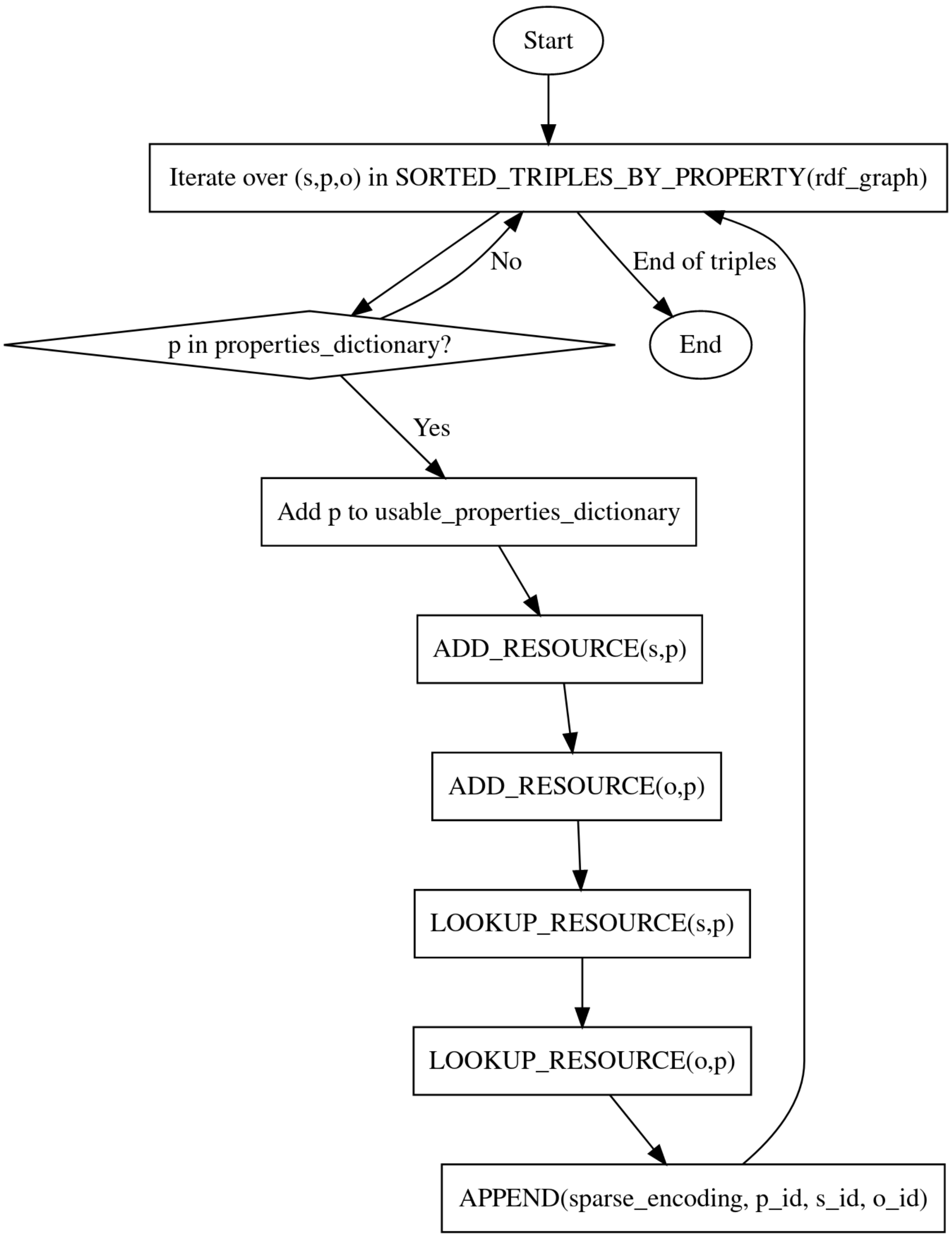

Start
Iterate over (s,p,o) in SORTED_TRIPLES_BY_PROPERTY(rdf_graph)
No
End of triples
p in properties_dictionary?
End
Yes
Add p to usable_properties_dictionary
ADD_RESOURCE(s,p)
ADD_RESOURCE(o,p)
LOOKUP_RESOURCE(s,p)
LOOKUP_RESOURCE(o,p)
APPEND(sparse_encoding, p_id, s_id, o_id)

Figure 7. Visual representation of the iteration algorithm.

2.3.2. ADD_RESOURCE Algorithm

The *ADD_RESOURCE* function is a critical component of the RDF graph processing procedure, which is designed to keep track of and assign unique identifiers to resources (subjects or objects) in the RDF graph. The function accepts a resource and a property as input. The property is used to determine the property group. An ID is assigned if the resource is already in the global or local resources dictionary. If not, it is added to the local resources dictionary and assigned an ID. The function ensures that all resources have unique IDs, whether globally or locally scoped[2].

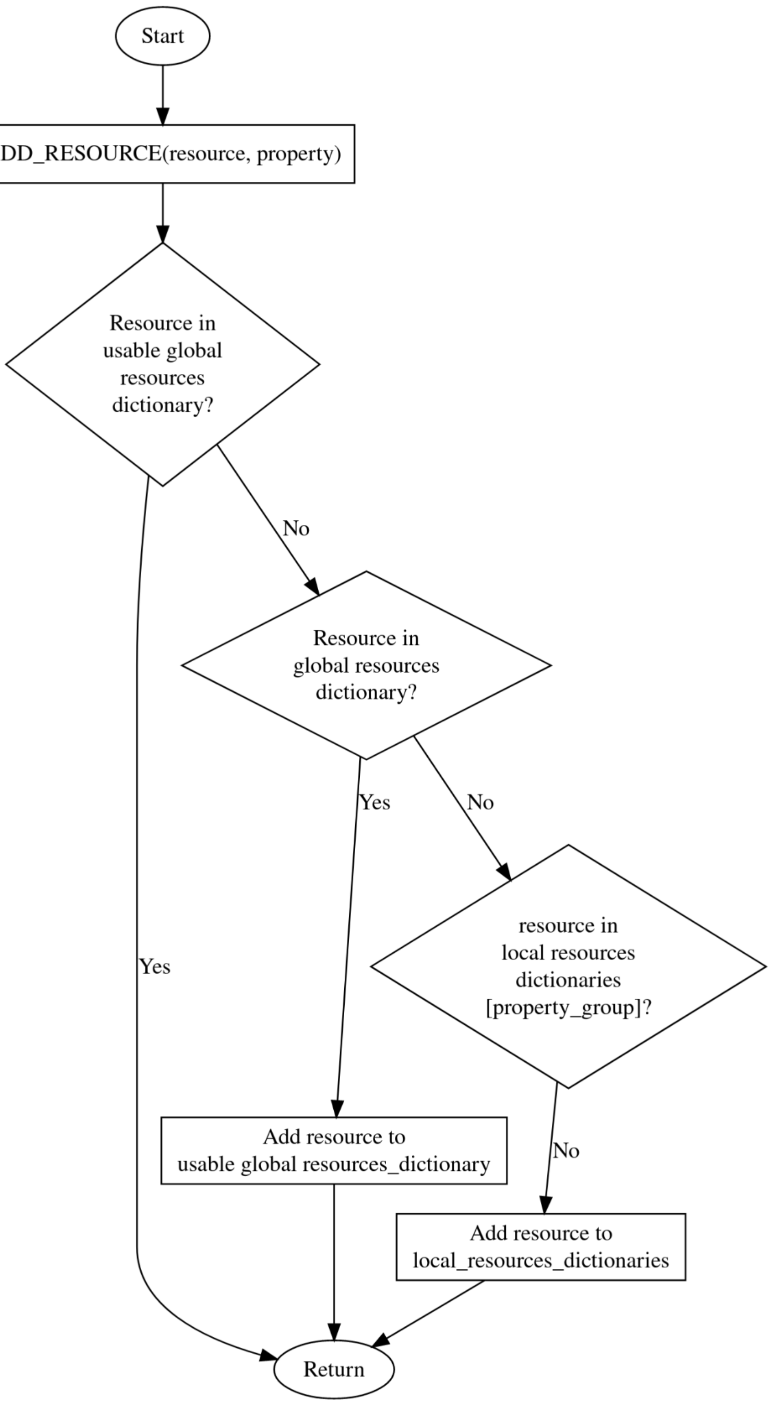

Start
ADD_RESOURCE(resource, property)
Resource in usable global resources dictionary?
No
Yes
Resource in global resources dictionary?
Yes
No
resource in local resources dictionaries [property_group]?
No
Add resource to usable global resources_dictionary
Add resource to local_resources_dictionaries
Return

Figure 8. Visual representation of the *ADD_RESOURCE* function.

2.3.4. LOOKUP_RESOURCE Algorithm

The *LOOKUP_RESOURCE* function retrieves a specific resource's unique identifier in the RDF graph. Given a resource and a property, the function determines the property group based on the property. Then, it checks if the resource is in the usable global resources dictionary. If it is, the function returns the ID of the resource from the usable global resources dictionary. If not, it checks if the resource is in the local resources dictionary for the property group. If it is, the function returns the ID of the resource from the local resources dictionary. The function raises an error and exits if the resource is not found in either dictionary. This function ensures that each resource in the RDF graph can be correctly and consistently identified during the graph processing procedure[2].

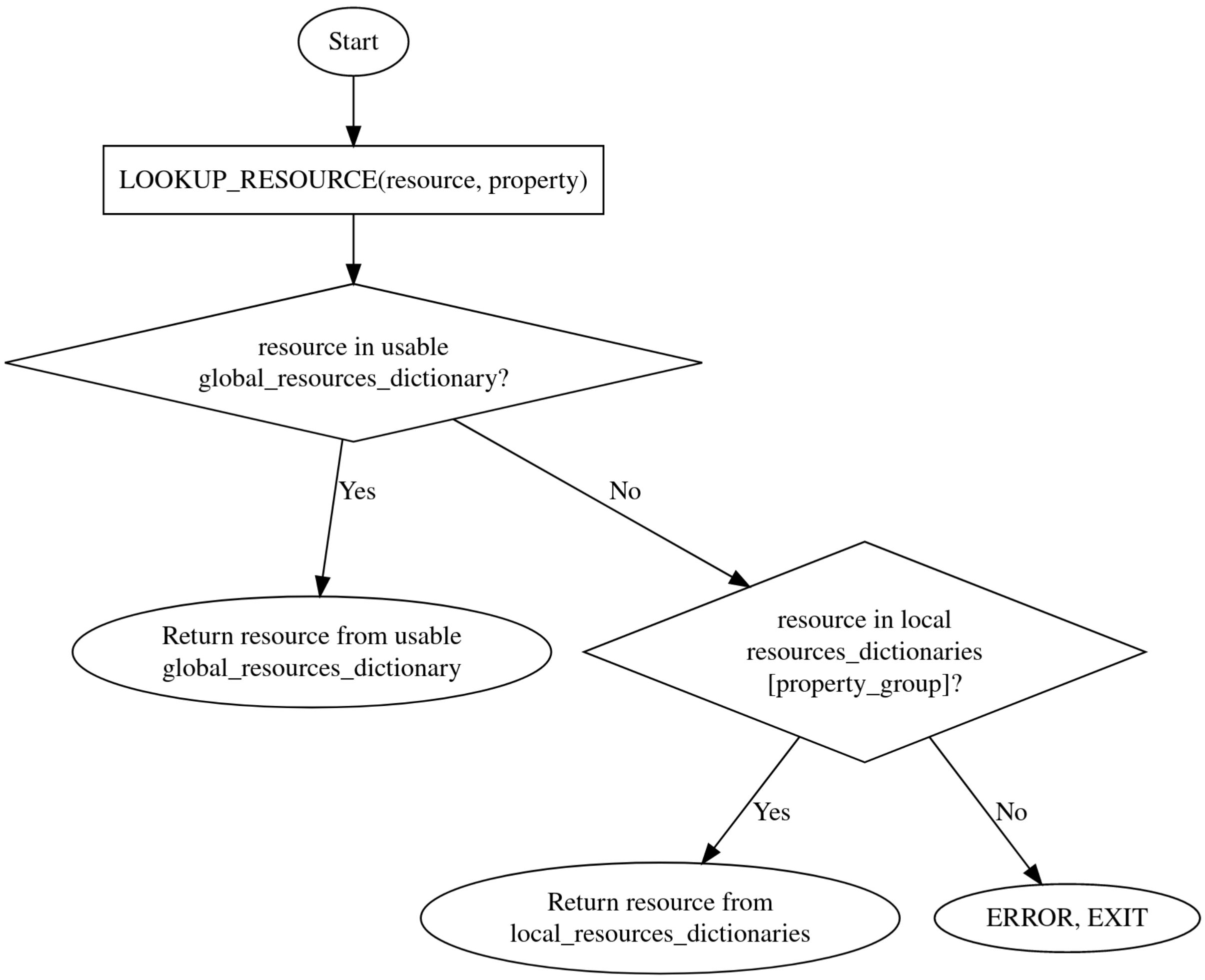

Start
LOOKUP_RESOURCE(resource, property)
resource in usable global_resources_dictionary?
Yes
No
Return resource from usable global_resources_dictionary
resource in local resources_dictionaries [property_group]?
Yes
No
Return resource from local_resources_dictionaries
ERROR, EXIT

Figure 9. Visual representation of the *LOOKUP_RESOURCE* algorithm.

2.3.5. Layering RDF Graphs and Encoding as 3D Adjacency Matrices

The first step in the visual representation of the deep learning-based approach involves layering RDF graphs and encoding them as 3D adjacency matrices [2]. This process is crucial for efficiently representing the graph structure and simplifying the subsequent reasoning process. Each graph layer corresponds to a specific aspect or property, which is then encoded in a 3D adjacency matrix [2]. The layers can be visualized as stacked 2D matrices, with each cell representing the relationship between a pair of nodes in the graph.

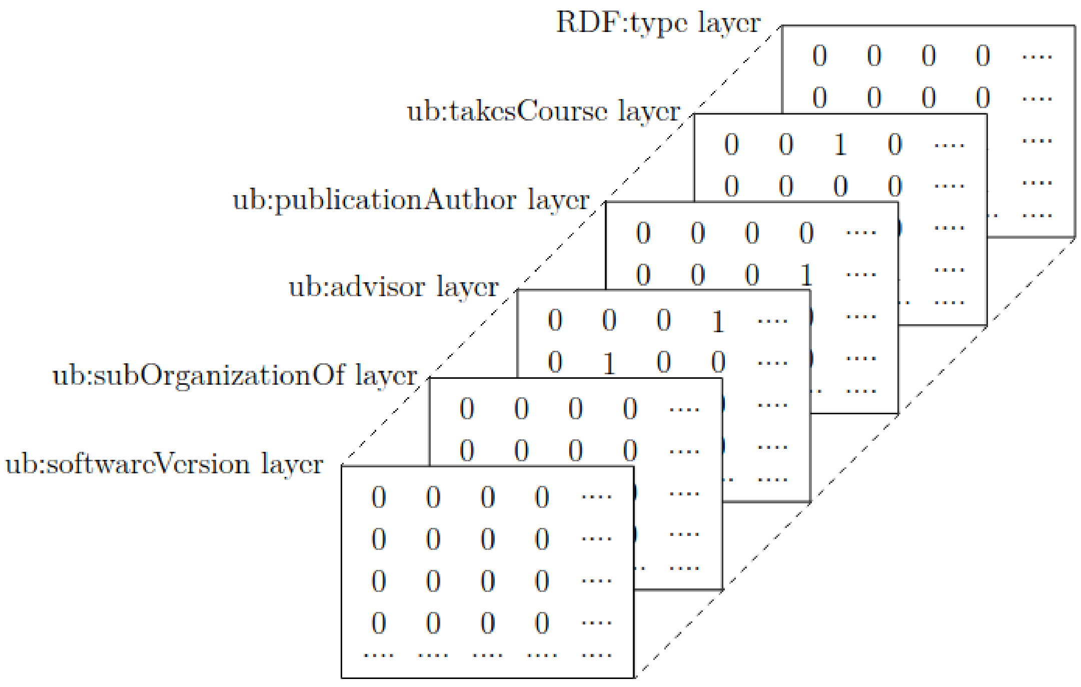


Figure 10. 3D Adjacency matrix (*tensor*) excerpt. [2]

2.3.6. Graph Word Formation and Sequences

Once the RDF graph has been layered and encoded as a 3D adjacency matrix, the next step is to form graph words and sequences. Graph words are formed by extracting substructures from the adjacency matrices, which can then be arranged into sequences representing the input graph and its entailments. These sequences serve as the input and output for the neural machine translation process and are crucial for understanding the relationships and entailments within the RDF graph.

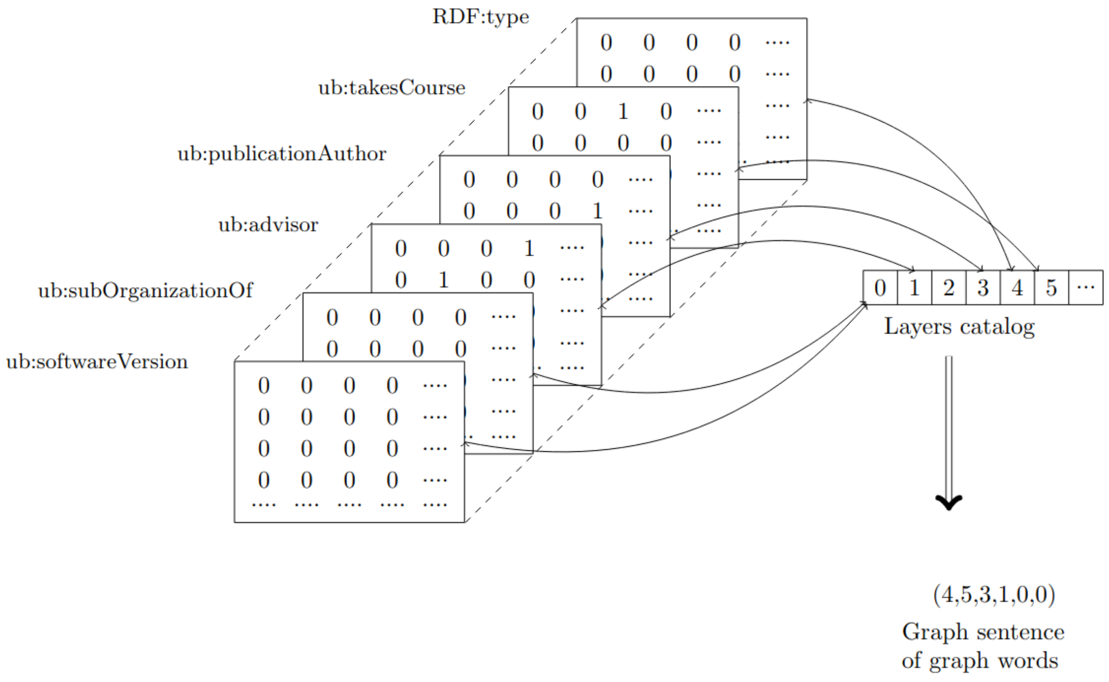


Figure 11. Converting the 3D adjacency matrix to a sentence of graph words [2].

2.3.7. RDFS Inference as Neural Machine Translation of Graph Word Sequences

The RDFS inference process in the deep learning-based approach can be visualized as the neural machine translation of graph word sequences[2]. The neural machine translation model takes the sequences of graph words as input and generates new sequences that represent the inferred relationships and entailments within the RDF graph. This translation process can be depicted as a series of transformations applied to the graph word sequences, ultimately leading to a more complete and accurate representation of the RDF graph[2].

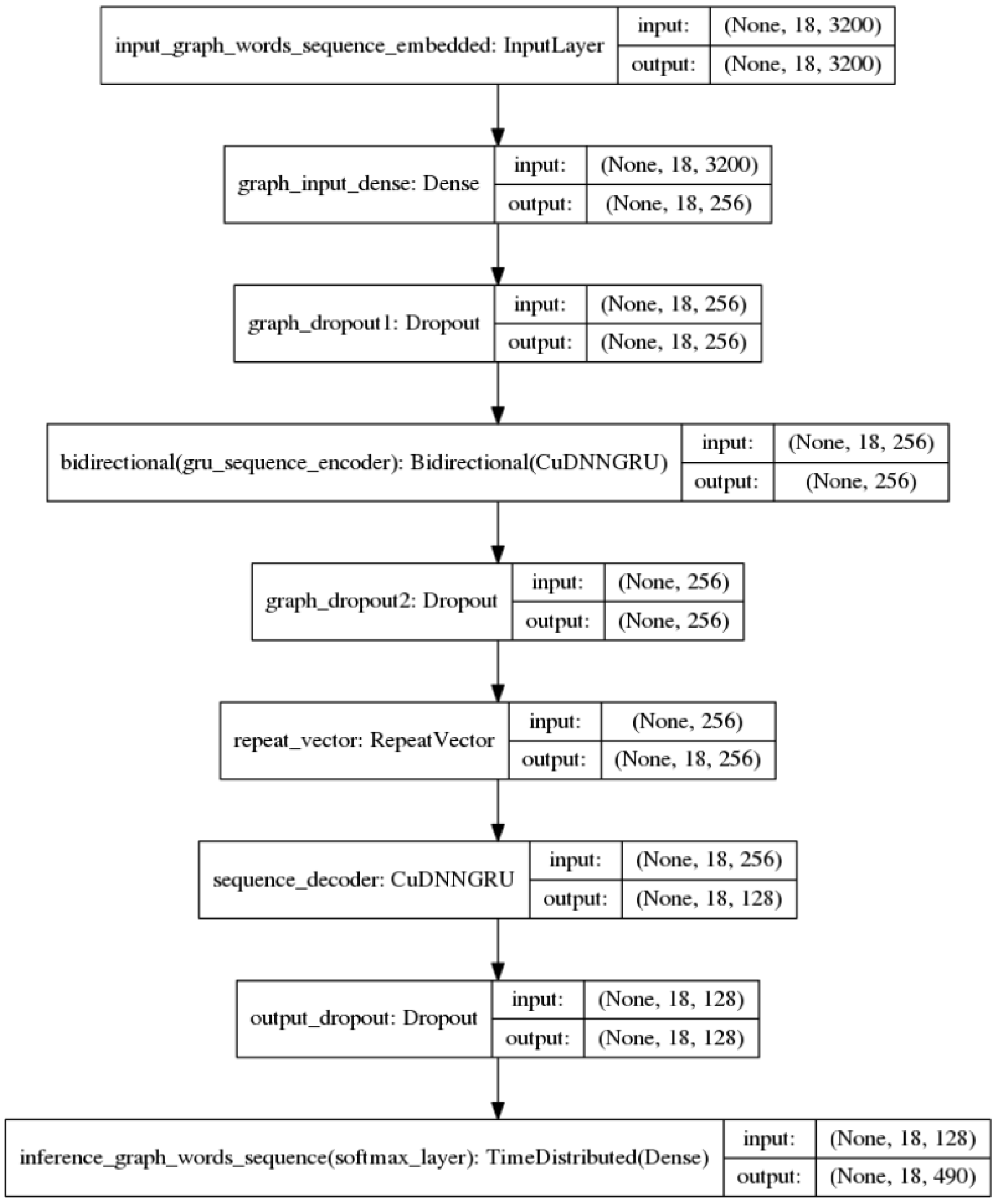


Figure 12. A visual representation of the RDFS inference process as neural machine translation of graph word sequences [2].

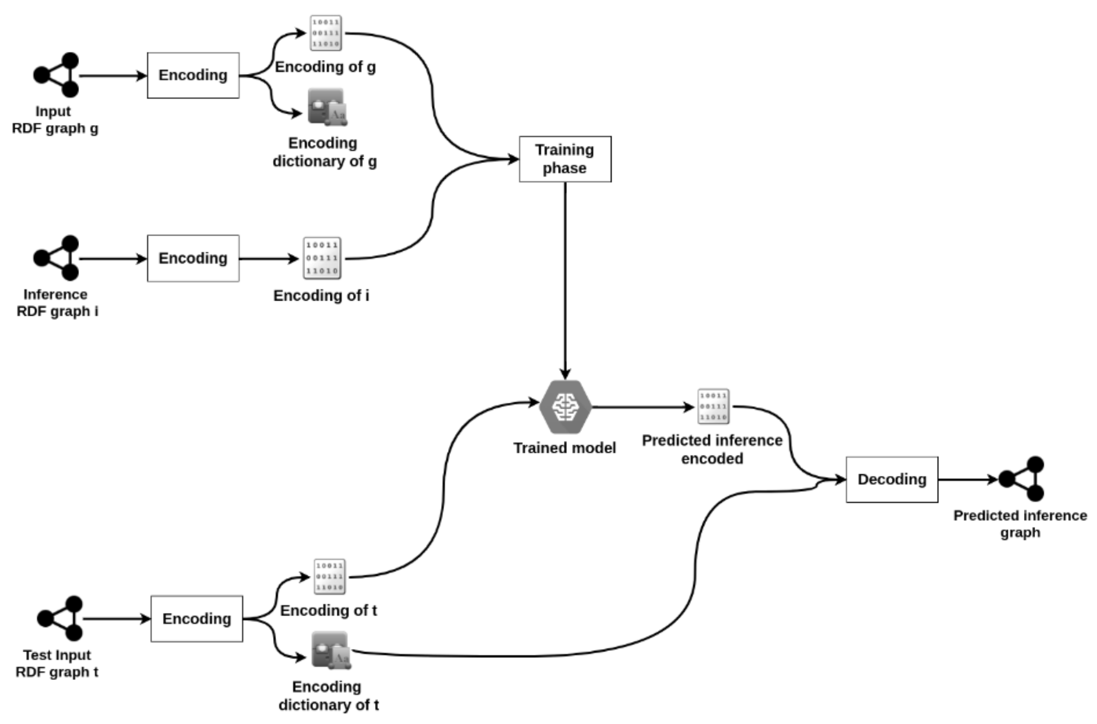


Figure 12. The stages of the deep learning-based RDFS materialization process.

2.4. Replication of the Deep Learning Paper

2.4.1. Goals and motivation

One of the objectives of this thesis is to deepen the understanding of the original research, verify its findings, and potentially discover avenues for process optimization. Another goal of the replication is to minimize the overall runtime of the process. It is posited that a significant fraction of the time complexity in the original study stems from the caching of intermediate data to disk. Such I/O operations can significantly hinder performance, mainly when dealing with large datasets [30], [36]. To mitigate this issue, a modification to the original algorithm is proposed whereby all data is retained in memory, thereby eliminating the need for disk caching. This adjustment, however, necessitates a thorough evaluation of the memory footprint and possible trade-offs in terms of computational efficiency [2], [16], [36].

The second goal of this replication is to gain an exhaustive understanding of the process. The intention is to dissect each step of the original study, critically examine the assumptions made, and thoroughly comprehend the mechanics of the algorithm. This comprehensive exploration is not only expected to enable a more accurate replication of the study, but it may also yield insights that could contribute to improving the process or its application to related problems [7].

The third goal of the replication is to understand the encoding system employed in the original research. Encoding is a fundamental component of numerous data processing tasks, transforming raw data into a format more amenable to specific analyses [2]. By scrutinizing the encoding system, the intention is to understand its design, efficacy, and potential limitations. This understanding could prompt suggestions for improvements or alternative encoding strategies.

2.4.2. Problems

In replicating the study, several unexpected challenges surfaced that added complexity to the replication process while providing significant learning opportunities. This section delineates the principal issues encountered and the strategies implemented to address them.

The encoding system utilized in the original research was intricate and had substantial implications for the study results [2]. However, it was observed that the system documentation needed more clarity and comprehensiveness. This lack impeded a complete understanding and accurate replication of the encoding processes. Extensive time and effort were invested in interpreting the available instances, and educated assumptions about the authors' intentions had to be made.

A considerable challenge was the discrepancy between the algorithm outlined in the academic paper[2] and the implementation provided in the corresponding GitHub repository [50]. This inconsistency bred confusion as attempts were made to reconcile the theoretical approach with the practical application. Often, it remained ambiguous as to which version of the algorithm was the intended, necessitating judgments based on an understanding of the overarching research objectives. An illustrative example of this issue is the implementation of the encoding algorithm from the repository.

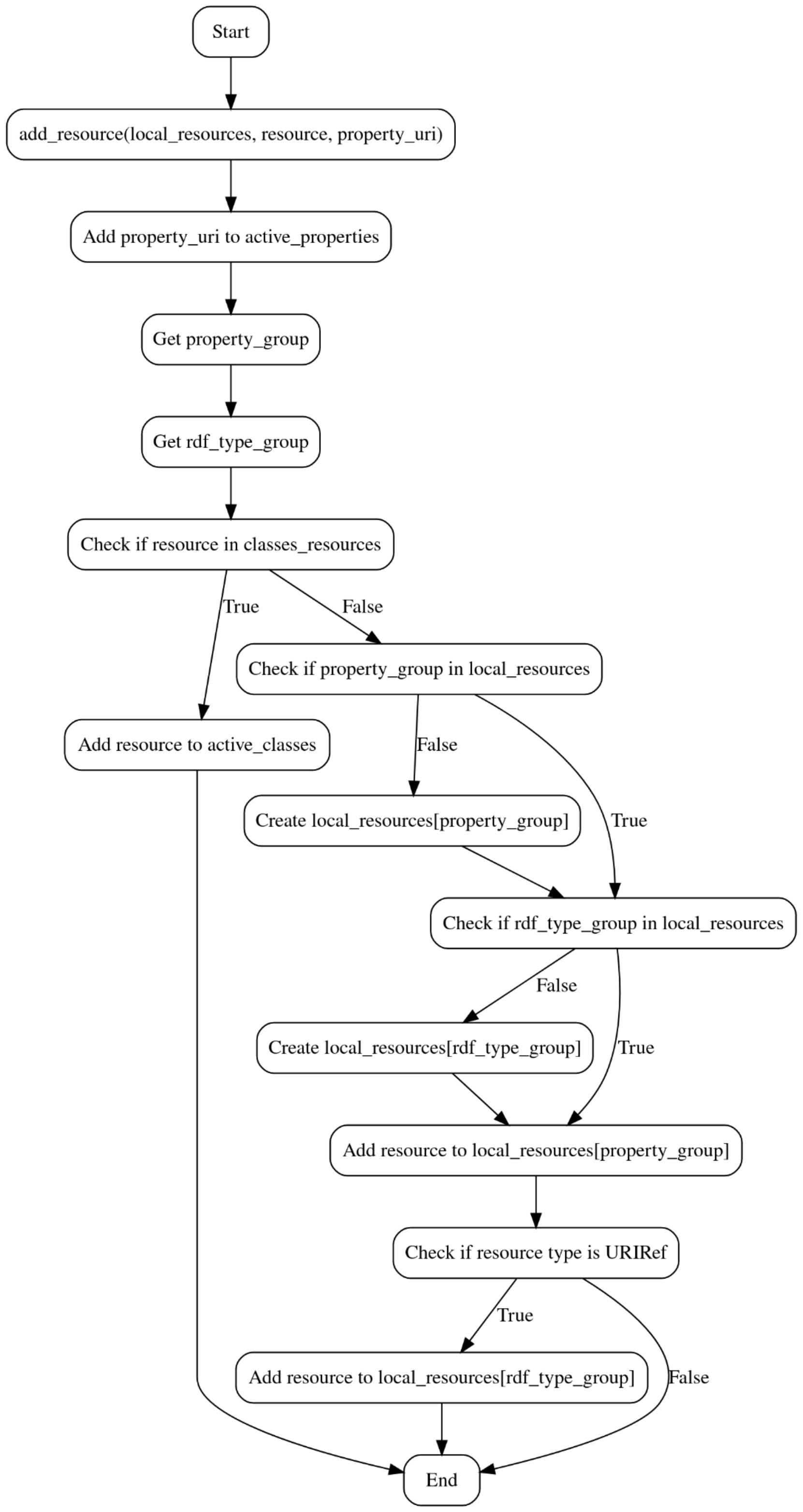

Start
add_resource(local_resources, resource, property_uri)
Add property_uri to active_properties
Get property_group
Get rdf_type_group
Check if resource in classes_resources
True
False
Check if property_group in local_resources
Add resource to active_classes
False
Create local_resources[property_group]
True
Check if rdf_type_group in local_resources
False
Create local_resources[rdf_type_group]
True
Add resource to local_resources[property_group]
Check if resource type is URIRef
True
Add resource to local_resources[rdf_type_group]
False
End

Figure 13. Process flow for the author's implementation of the *ADD_RESOURCE* algorithm. [50]

An additional obstacle emerged in relation to SPARQL queries, which were employed for graph generation and inference in the original study. Documentation for these queries was scant, making comprehension of their structure and function reliant on an in-depth knowledge of the SPARQL query language and the specific dataset in use. Significant effort was expended in mastering SPARQL and experimenting with diverse queries to understand their impact on graph generation and inference processes.

Further complications arose due to system-dependent functions, such as the SPARQL DESCRIBE function [30] and the file structure. These elements of the original code were tailored to the specific system on which the initial research was conducted, resulting in compatibility issues with the replication system. To resolve this, the code had to be altered to fit the configuration of the replication system, necessitating a comprehensive understanding of both the original and the replication systems.

Finally, discrepancies were identified between the TBOX and ABOX utilized by the original author and those available from the LUBM dataset. These differences affected the thesis' outcomes, and reconciling these inconsistencies proved challenging. Time had to be invested in examining the LUBM dataset and the author's unique application of it to comprehend the implications of these variances.

2.4.3. Results

Initial attempts to replicate the encoding classes outlined in the original study were unsuccessful, primarily due to insufficient documentation and disparities between the research paper and its corresponding GitHub repository [30]. Rather than attempting to reconstruct the precise encoding classes, the decision was made to utilize those already in existence, despite their differences. This course of action allowed for a greater focus on the overarching principles of encoding and their implications on the results of this thesis.

The replication of the graph inference algorithm presented notable challenges. The paper did not document the algorithm, necessitating exploring alternative methods for understanding and implementation. Initial efforts centered on manually stating the logic based on available information. While this provided some insight, it did not yield the anticipated results due to the algorithm's complexity and the scarcity of detail. Subsequent efforts involved the Jena inference engine, briefly referenced in the original paper [30].

While the Jena engine facilitated some degree of inference, it fell short of fully replicating the original algorithm. The discrepancies observed in the results underscored the intricate nature of the inference process and the influence of various factors on the outcomes.

Example 14: Triples from the *<http://www.Department0.University0.edu /AssistantProfessor0/Publication0>* graph, provided in the GitHub repository by the study's author.

For convenient reading the prefixes have been replaced as follows:

```
http://www.Department0.University0.edu as "Uni0Dep0"
http://www.w3.org/2000/01/rdf-schema as "rdfs:"
http://www.w3.org/1999/02/22-rdf-syntax-ns as "rdf:"
http://www.Department0.University0.edu" as "Uni0Dep0"
http://swat.cse.lehigh.edu/onto/univ-bench.owl#" as "ub:"
AssistantProfessor as “AProf”
GraduateStudent as “GrStud”
```

```
<Uni0Dep0/AProf.0/Publication0> <ub:publicationAuthor> <Uni0Dep0/GStud.113>
<Uni0Dep0/AProf.0/Publication0> <rdf:type> <ub:Publication>
<Uni0Dep0/AProf.0/Publication0> <ub:publicationAuthor> <Uni0Dep0/AProf.0>
<Uni0Dep0/AProf.0/Publication0> <ub:publicationAuthor> <Uni0Dep0/GStud.44>
<Uni0Dep0/AProf.0/Publication0> <ub:name> "Publication0"
```

These graph triples were supplied in the GitHub repository by the original author of the paper [30]. The generation process of these triples was described as utilizing the SPARQL DESCRIBE function, which is contingent upon the system and RDF store in use. This process was successfully replicated using Python's RDF library and Apache Jena Fuseki SPARQL graph database engine[2]. The initial approach entailed searching for triples where the graph name was present in the "subject" or "object" fields. Subsequently, the replication in SPARQL involved generating the immediate 1-hop neighborhood via a specific query.

Example 15: SPARQL query for finding the 1-hop neighborhood:

```
CONSTRUCT {
  ?s ?p ?o .
}
WHERE {
  { <URI> ?p ?o . BIND(<URI> AS ?s) }
  UNION
  { ?s ?p <URI> . BIND(<URI> AS ?o) }
}
```

The resulting set of triples can be represented visually as a graph. This representation is beneficial in visually discerning the relationships and linkages between the different entities in the graph.

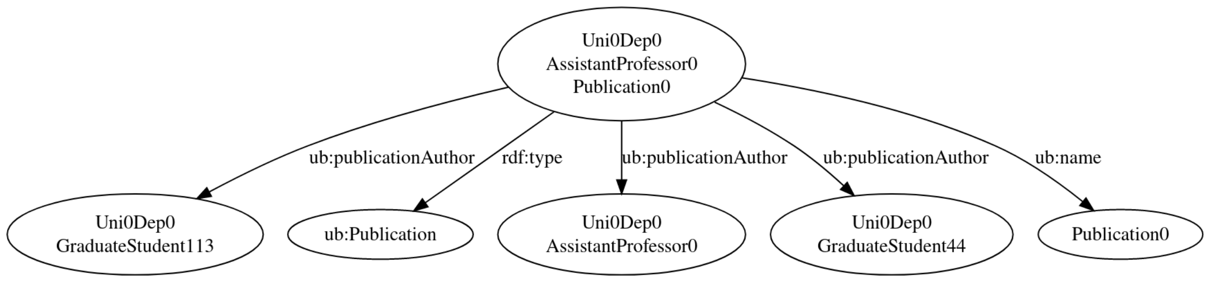


Figure 14. Visual graph representation of the un-materialized *Publication0* graph.

The collection of triples, as inferred by the original author of the paper, forms another key component of the study. These inferred triples shed light on the RDF graph's implicit information and underlying structure, demonstrating the power and utility of semantic reasoning in knowledge extraction.

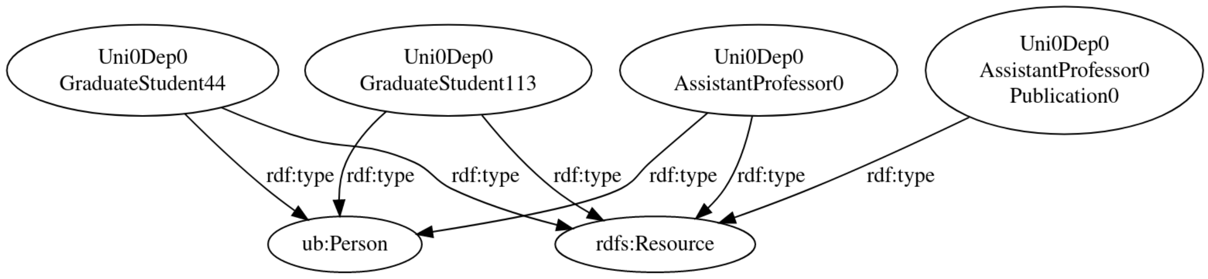


Figure 15. Visual graph representation of the inferred triples of the *Publication0* graph.

By combining the original *input* graph with the graph of *inferred* triples, a *materialized* graph is produced. This graph encompasses all potential inferences, thus rendering all relationships within the dataset explicit. A materialized graph provides a comprehensive perspective of the dataset's interconnectedness. This fully materialized graph augments the efficiency of querying and analysis procedures by transforming all implicit knowledge into explicit representations.

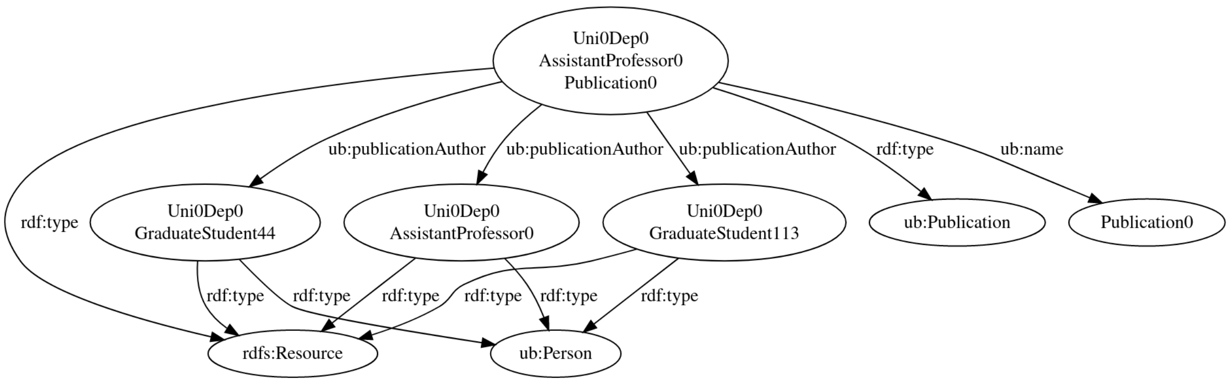


Figure 16. Visual graph representation of the materialized *Publication0* graph.

The final materialized graph, as provided by the original study's author, comprises 13 triples. However, replication of the same proved to be unsuccessful. The 2-hop neighborhood within the un-materialized ABOX and TBOX comprised 44 triples (A visualization of the graph is presented in Appendix A.), and this number rose to 52 triples in the fully materialized ABOX and TBOX. Another methodology attempted involved retaining the highest level RDFS descriptions (i.e., those not a subclass of any other Class apart from RDFS base classes). This method also did not yield success, as the author's inference did not consistently refer to the topmost Class node but frequently to the second or third [30].

The initial paper does not explain the materialization process in-depth, merely stating that the Java Jena inference engine was employed [2]. The replication study successfully executed the encoding and decoding on the extant dataset. This operation served as an affirmation that the fundamental comprehension of the encoding and decoding procedures was essentially accurate, irrespective of the inability to replicate the precise classes accurately.

The final phase of the replication was dedicated to the training and inference of data based on the existing dataset. The results revealed that both the training and inference processes functioned effectively, further substantiating the understanding of the encoding and the graph inference processes. These findings were promising, demonstrating that meaningful outcomes could still be achieved.

3. Evaluation

3.1. Evaluation Metrics and Methodology

Evaluation metrics and methodology form the cornerstone of any research or experiment that necessitates a comparative analysis of distinct techniques or methodologies. In contrasting Datalog-based and machine learning-based strategies for RDFS materialization and link prediction assignments, selecting relevant evaluation metrics and a solid methodology is critical for ensuring the credibility and dependability of the outcomes. Several evaluation metrics can be utilized [51] to assess the effectiveness of Datalog-based and machine learning-based approaches. Accuracy is the ratio of accurately inferred edges to the total edges considered; this metric assesses the methods' proficiency in predicting the correct linkages amongst nodes in the RDF graph [2].

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$ [2]

Another method is comparing the ratio of True Positives (TP), True Negatives(TN), False Positives(FP), and False Negatives(FP).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$ [2]

Completeness is the metric that signifies the ratio of inferred edges to the total possible edges in the RDF graph, reflecting the methods' capacity to generate all potential inferences based on the RDFS schema.

$$\text{Completeness} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$ [2]

Consistency refers to the degree of conformity of the materialized RDF graph to the RDFS schema and its constraints, thus evaluating the methods' capability to generate logically consistent inferences.

By denoting V as the total number of inferences in the materialized RDF graph, C as the number of these inferences that conform to the RDFS schema and its constraints :

$$\text{Consistency (Cons)} = \frac{C}{V}$$ [2]

For measuring computational efficiency, the processing time needed to materialize the RDF graph, measurable in terms of CPU time or wall-clock time, serves as a metric to contrast the methods' performance relative to processing speed. A possible metric for this would be dividing the number of inferences (*N*) in the materialized RDF graph over the total processing time *(T)*.

$$\text{Computational Efficiency (E)} = \frac{N}{T}$$ [51]

For the deep learning method, an even more detailed approach was employed. Each phase of the process was individually timed, from the initial encoding stage, through the generation of three-dimensional graph layers, to the production of graph words, and finally to the training stage. This approach facilitated a thorough examination of the time requirements at each stage, thereby providing a complete picture of the overall computational efficiency of the deep learning method.

The memory usage of the Datalog-based method was assessed as a whole during the materialization process. This method's memory requirements provided a benchmark for comparison, shedding light on the relative efficiency of the two methods to memory utilization.

3.2. Dataset and Experimental Setup

Selecting a suitable dataset and an appropriate experimental setup constitute key steps in comparing the performance of Datalog-based and machine learning-based strategies for RDFS materialization. In this regard, the Lehigh University Benchmark (LUBM) ontologies are a fitting choice given their detailed design for evaluating semantic web technologies [4].

LUBM provides a variety of datasets, such as LUBM1, LUBM2, LUBM5, and LUBM10, each signifying distinct sizes and complexities of RDF graphs. These datasets encompass synthetic data about universities, departments, students, and courses, among other related entities. These are generated in line with a predefined ontology and schema. By employing these datasets, researchers are empowered to probe the scalability and efficiency of the Datalog-based and machine learning-based methodologies across a spectrum of data volumes and complexities.

3.2.1. Preprocessing

Ensuring the RDF graphs encapsulated within the LUBM datasets were correctly formatted and compatible with both the Datalog-based and machine learning-based methods was most important. A discrepancy was noted during the thesis between the URI prefixes in the initial TBOX and those in the subsequently generated one. This difference posed a potential issue for consistency and comparability of results across different process stages. It underscored the importance of maintaining uniform naming conventions and consistent identifiers throughout the lifecycle of the dataset. Careful attention to such details is crucial for the reproducibility and robustness of the analysis, as discrepancies in URI prefixes can influence the interpretation of results and the ability to draw accurate conclusions.

3.2.2. Parameter tuning:

To replicate the original study as closely as possible, the initial parameters were consistent with the original experiment's. This decision was premised on the objective of ensuring a high degree of fidelity in the replication process[2]. Therefore, no hyperparameter optimization was undertaken, despite its potential for enhancing performance. This is because the primary goal was not to surpass the initial results but to understand and validate the original findings. Thus, any modifications to the initial parameters, such as hyperparameter optimization, were considered outside the scope of this thesis.

3.2.3. Hardware and software configurations

The initial experiment used a Unix filesystem as the base for the original study[2]. However, this codebase proved incompatible with a Windows 10 computer without employing Linux subsystems. Initially, an attempt was made to execute the experiment on a Windows 10 platform, but it eventually became necessary to switch to Ubuntu due to these compatibility issues. Regrettably, some unforeseen memory issues caused the Ubuntu operating system to fail. As a result, a transition to a Linux Mint operating system was necessitated, and the experiment was successfully conducted. This transition illustrates the critical role that hardware and software configurations can play in conducting and replicating computational experiments.

The system information was as follows:

Operating System: Linux Mint 21.1 Vera, based on Ubuntu 22.04 jammy, with Kernel version 5.15.0-56-generic for 64-bit architecture.
Desktop Environment: Cinnamon 5.6.5, with GTK 3.24.33 and Window Manager Muffin.
Machine: A Desktop with a Micro-Star B450 PLUS MAX (MS-7B86) motherboard, UEFI by American Megatrends LLC. (version H.G0).
CPU: AMD Ryzen 7 2700X, an 8-core processor with 64-bit architecture, a Zen+ architecture, and a speed varying between 1887 MHz and 2719 MHz.
Graphics: NVIDIA GeForce GTX 1070, powered by a Micro-Star MSI driver. The display was managed by X.Org v1.21.1.3 with a loaded NVIDIA driver.
Network: Realtek RTL8111/8168/8411 PCI Express Gigabit Ethernet.
Drives: Kingston SA400S37240G (223.57 GiB) and a Samsung SSD 860 EVO 500GB (465.76 GiB).
Partition: The main partition (/) was 218.51 GiB, out of which 196.07 GiB was used.
Repos: The system had 2585 packages, 2574 from apt and 11 from flatpak.

During the experimental studies, the evaluation and comparison of memory costs and runtime between the Datalog-based and machine learning-based methods were deemed imperative. Memory costs were assessed by continuously monitoring memory usage throughout the materialization process, while runtime was evaluated by measuring the period required to materialize the RDF graphs. By examining the memory costs and runtime across the ontologies (datasets) LUBM1, LUBM2, LUBM5, and LUBM10, it was possible to gain an understanding of the scalability and efficiency of the Datalog-based and machine learning-based methods within the confines of RDFS materialization and link prediction tasks. This analysis pinpointed each method's potential advantages and constraints, providing a foundation for further exploration.

3.3. Memory Costs and Runtime for Datalog

To evaluate the performance of the deep learning method, the first step was to evaluate the materialization performance of a traditional reasoner using Datalog and compare it with the deep learning approach. This was executed to determine the two methodologies' relative efficiencies and operational effectiveness.

The results demonstrated the efficacy of a reasoner, chibi when executed on diverse datasets (lubm1, lubm2, lubm5, lubm10, lubm20, lubm50) in a parallel computing environment. Each dataset was subjected to a sequence of processing phases: initial materialization, positive and negative updates. The preliminary materialization phase entailed deriving new triples from pre-existing ones. In contrast, positive updates implied the augmentation of inferred triples, and negative updates signified the elimination of no longer valid triples.

The batch size, indicative of the count of triples processed concurrently, escalated in conjunction with the dataset size, as was mirrored in the inference time. The number of triples processed and the inference time, quantified in seconds, exhibited a direct relationship with the batch size and the number of triples, suggesting that the process duration extended with the enlargement of the dataset. Interestingly, despite the escalation in triples and inference time across datasets, the number of triples retained consistency across different phases within a singular dataset. This insinuated that neither were new triples inferred nor were any triples discarded during the positive and negative update stages.

Table 1. Materialization results using a Datalog reasoner on various LUBM datasets.

Dataset	*Batch Size*	*Inference Time (s)*	*Number of Triples*	*Memory Used (KB)*
lubm1	103391	0.87	145051	368448
lubm2	237461	2.09	329221	855412
lubm5	645974	5.91	889929	2251048
lubm10	1316657	12.74	1811142	4638192
lubm20	2781677	28.83	3823801	9779320
lubm50	6889057	79.53	9464860	23894516

The analysis revealed an intriguing pattern regarding dataset size, the quantity of inferred triples, and memory utilization. Despite the dataset size demonstrating a near-linear escalation (approximately 2 to 2.5 times) and the inferred triples also presenting a small growth (approximately 1.3 times the input), the memory consumption illustrated an exponential surge across the datasets.

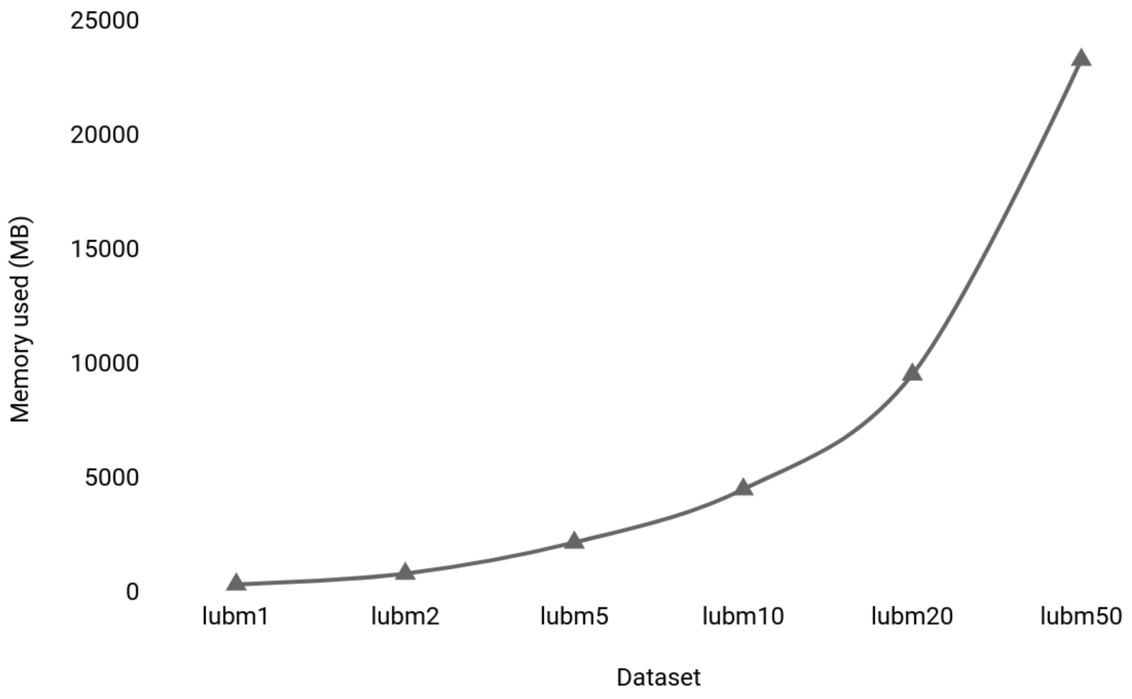


Figure 17. Memory usage over datasets.

It was also observed that memory usage correlated linearly with inference time. This suggests that as the dataset size and the inference complexity grow, the memory requirements disproportionately amplify, resulting in an exponential memory footprint. This raises significant concerns about scalability and performance, especially when dealing with larger datasets and more complex inferences.

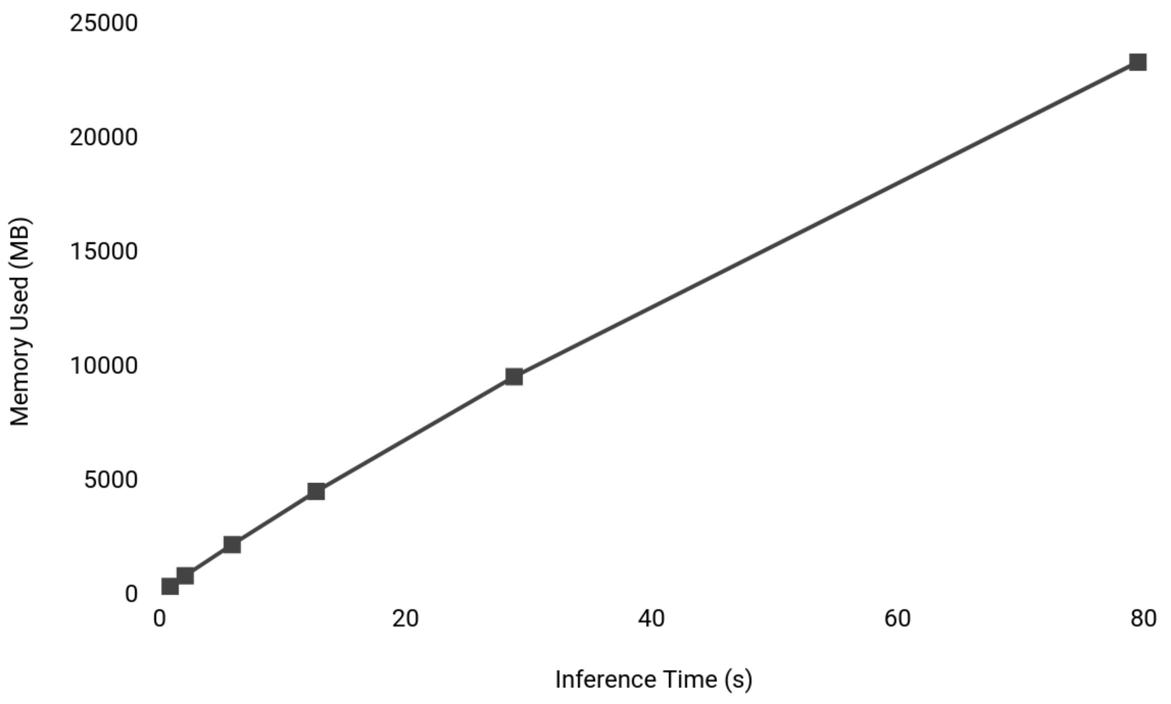


Figure 18. inference time compared to the memory used.

3.4. Memory Costs and Runtime for Deep Learning

To ensure the generalizability of the findings, the entire process was replicated ten times, except for training the model trained on both the CPU and GPU just once. This strategy facilitated mitigating anomalous results that could distort the outcome due to chance variables. Therefore, the performance measures obtained from each run were averaged, providing a more representative and reliable estimate of the metrics.

The model training for the materialization of the RDF graph was partitioned into ten progressive phases to monitor the distinct stages of the operation. The first stage involved loading the data from the un-materialized ABOX and TBOX files and preparing it for further processing. The graph creation took approximately 5.04 seconds and used 140.56 MiB of memory. Filtering BNodes consumed about 1.99 seconds and 83.46 MiB of memory.

Generating 1-hop sub-graphs required 1.88 seconds and 54.09 MiB of memory, while processing the ABOX dataframe took 5.47 seconds and used 33.86 MiB. The total time taken for this step was approximately 14.38 seconds, and the memory used was approximately 1379.74 MiB. In the second step, dictionaries were created. Creating the properties dictionary took 0.15 seconds and used 0.14 MiB of memory. Meanwhile, the resources dictionary was created in 0.02 seconds without significant memory usage, and the sub-properties dictionary was created in 0.09 seconds, using 0.05 MiB of memory.

Table 2. Mean time and memory usage for data preparation and model creation.

Step	*Task*	*Time Taken (seconds)*	*Memory Used (MiB)*
1	Graph Creation	5.04	140.56
1	Filtering Bnodes	1.99	83.46
1	Generating 1-hop Sub-graphs	1.88	54.09
1	Processing the Whole ABOX Dataframe	5.47	33.86
2	Creating Properties Dictionary	0.15	0.14
2	Creating Resources Dictionary	0.02	0
2	Creating Subproperties Dictionary	0.09	0.05
3	Creating Graph Catalogue	19.05	538.40
4	Encoding the Graph	11.34	39.28
5	Creating Graph Words	0.29	8.77
6	Creating Matrix Embedding	18.91	89.11
7	Training-Test Split	0.05	0.32
8	Creating Graph Words Encoder, Graphs, and Model	3.14	7730.86
9	Training the Model (CPU)	5549.03	19892.32
9	Training the Model (GPU)	2193.40	-
10	Evaluating the Model	94.16	1038.75

The third step involved creating a catalog, where the graph words encoder was created, and graphs were generated from the read files. This process took about 19.05 seconds and used 538.40 MiB of memory. The fourth step involved encoding the graph, which took approximately 11.34 seconds and used 39.28 MiB of memory.

In the fifth step, the graph words were created in 0.29 seconds, and the memory used for this process was 8.77 MiB. The sixth step was to create matrix embeddings, which took about 18.91 seconds and used 89.11 MiB of memory. In the seventh step, the dataset was split into training, validation, and test sets in the ratio of 60:20:20, respectively. The process took 0.05 seconds and used 0.32 MiB of memory. The eighth step was to create the graph words encoder, graphs, and model. This step took approximately 3.14 seconds and used a significantly larger memory chunk of 7730.86 MiB. The ninth step involved training the model for 200 epochs with a batch size of 128. The model training took a considerable amount of time, approximately 5549.03 seconds. The final step was to evaluate the model on the test set, which took about 94.16 seconds and used 1038.75 MiB of memory. The model achieved a test set accuracy of 0.9881, indicating that the model was able to correctly predict the class of the instances about 99% of the time.

The table illustrates a significant improvement in speed from Step 1 (graph creation) to Step 3 (creating graph words encoder and graphs). While the graph creation process in Step 1 took around 5.04 seconds, the process in Step 3 was approximately four times faster, taking about 19.05 seconds. This speed increase could improve overall system efficiency and throughput.

A high memory usage was observed during the creation of the Graph Words Encoder, Graphs, and Model in Step 8, consuming approximately 7730.86 MiB. This step is significantly more memory-intensive than others, which may require more efficient memory management or higher capacity hardware for large-scale applications. The model training process (Step 9) is the most time-consuming, taking about 5549.03 seconds or 1.5 hours. Training the model on the GPU was around 2.52 times faster or around 2193.40 seconds. This indicates that while other steps have been optimized for speed, model training remains a computationally intensive process that may benefit from additional optimization efforts. The model evaluation in Step 10 is also quite time-consuming, taking 94.16 seconds. This is a critical step to measure the model's performance, and this time might increase with the model's complexity and the test dataset's size.

The model achieved an accuracy of approximately 0.9881 on the test set. While this is a good result, there may be room for further model improvement or tuning to achieve higher accuracy. While the results show a promising speed increase in graph creation and processing, the lack of documented inference during these steps limits the full utilization of these methods. Future work should focus on improving the transparency and traceability of the processes to leverage their potential fully.

The evaluation of the inference process for a single graph revealed a period ranging from 1 to 30 seconds, which, while seemingly inconsequential in isolation, amplifies significantly when dealing with larger data sets. For instance, in a scenario with over 20,000 graphs, this time consumption rapidly escalates, rendering the process untenably slow. Consequently, in terms of efficiency, the graph reasoner falls markedly short compared to its Datalog counterpart. The differential efficiency between the two underlines the challenges that persist in optimizing the performance of graph reasoners for extensive, complex datasets and emphasizes the ongoing relevance of traditional Datalog reasoning methods in specific contexts.

4. Conclusion

Graph Neural Networks (GNNs), a branch of deep learning methods explicitly designed for graph-structured data, have arisen as potent instruments for addressing the challenges inherent in the scalability of machine learning-based link prediction techniques, especially for large-scale graphs[2], [48]. GNNs can discern complex patterns and relationships between nodes by integrating local and global information, using iterative message-passing mechanisms to update node embeddings and generate representations encapsulating local and global graph structures. However, these methods have challenges, including the interpretability of models like GNNs. Discerning the reasoning behind these models' predictions is vital for building trust and ensuring their applicability in real-world scenarios. Despite this, a deep learning-based approach for RDFS materialization could potentially overcome various issues associated with traditional methods, such as computational and storage demands, handling of noisy or incomplete data handling, and scalability[2]. By leveraging GNNs, more robust and efficient techniques for RDFS materialization could be developed, capable of managing large, complex, and dynamic datasets.

Traditional RDFS reasoners emphasize soundness and completeness, presuming that the incoming data is precise and devoid of discrepancies. However, this presumption often fails to align with real-world situations where data might be partial, inconsistent, or contradictory [9]. These reasoners typically need help with such unclean data, resulting in imprecise or incomplete materialization outcomes. The task, therefore, is to devise reasoning methodologies that can effectively deal with the inherent discrepancies in web data without compromising soundness and completeness [2], [8]. The Web of data is intrinsically fraught with discrepancies due to various factors like data entry mistakes, inconsistencies in data sources, and the volatile nature of web data. This noise presents significant hurdles for traditional RDFS reasoning techniques, which are constructed to function based on the premise of accurate input data [2], [3]. The existence of noise can precipitate incorrect inferences, affecting the overall quality and dependability of the materialization process. Overcoming this challenge calls for the evolution of noise-resilient reasoning techniques to manage incomplete and inconsistent data while preserving the required degree of soundness and completeness [2].

In assessing the inference process, a single graph took between 1 to 30 seconds, a timeframe that becomes problematic when handling larger datasets of over 20,000 graphs. The graph reasoner, therefore, lagged significantly in efficiency compared to its Datalog counterpart. This highlights the optimization challenges for graph reasoners with extensive datasets and underscores the continued relevance of traditional Datalog methods in specific scenarios.

The research also highlighted the importance of clear and comprehensive documentation of the encoding system used in the study, which was found to be lacking, inhibiting complete understanding and accurate replication of the encoding processes. Additionally, inconsistencies between the algorithm detailed in the academic paper and the implementation provided in the corresponding GitHub repository further complicated the replication process, often leaving the intended version of the algorithm unclear. The replication of the graph inference algorithm presented significant challenges due to its complexity and the need for more detailed documentation. Despite efforts to manually state the logic and use the Jena inference engine, the results fell short of fully replicating the original algorithm, highlighting the intricacies of the inference process and the influential role of various factors in achieving the outcomes.

References

[1] A. Polleres, A. Hogan, R. Delbru, and J. Umbrich, “RDFS and OWL reasoning for linked data,” in *Reasoning web. semantic technologies for intelligent data access: 9th international summer school 2013, mannheim, germany, july 30 – august 2, 2013. proceedings*, vol. 8067, S. Rudolph, G. Gottlob, I. Horrocks, and F. van Harmelen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 91–149.

[2] B. Makni and J. Hendler, “Deep learning for noise-tolerant RDFS reasoning1,” *SW*, vol. 10, no. 5, pp. 823–862, Sep. 2019, doi: 10.3233/SW-190363.

[3] D. Tomaszuk and D. Hyland-Wood, “RDF 1.1: knowledge representation and data integration language for the web,” *Symmetry*, vol. 12, no. 1, p. 84, Jan. 2020, doi: 10.3390/sym12010084.

[4] “Lehigh University Benchmark (LUBM).” http://swat.cse.lehigh.edu/projects/lubm/ (accessed May 17, 2023).

[5] M. A. Khamis, H. Q. Ngo, R. Pichler, D. Suciu, and Y. Remy Wang, “Datalog in Wonderland,” *SIGMOD Rec.*, vol. 51, no. 2, pp. 6–17, Jul. 2022, doi: 10.1145/3552490.3552492.

[6] T. Ajileye, B. Motik, and I. Horrocks, “Streaming partitioning of RDF graphs for datalog reasoning,” in *The semantic web: 18th international conference, ESWC 2021, virtual event, june 6–10, 2021, proceedings*, vol. 12731, R. Verborgh, K. Hose, H. Paulheim, P.-A. Champin, M. Maleshkova, O. Corcho, P. Ristoski, and M. Alam, Eds. Cham: Springer International Publishing, 2021, pp. 3–22.

[7] C. F. Draschner, C. Stadler, F. Bakhshandegan Moghaddam, J. Lehmann, and H. Jabeen, “DistRDF2ML - Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs,” in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, New York, NY, USA, Oct. 2021, pp. 4465–4474, doi: 10.1145/3459637.3481999.

[8] H. T. Lin, N. Koul, and V. Honavar, “Learning Relational Bayesian Classifiers from RDF Data,” in *The semantic web – ISWC 2011*, vol. 7031, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, and E. Blomqvist, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 389–404.

[9] U. Lösch, S. Bloehdorn, and A. Rettinger, “Graph kernels for RDF data,” in *The Semantic Web: Research and Applications: 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, vol. 7295, E. Simperl, P. Cimiano, A. Polleres, O. Corcho, and V. Presutti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 134–148.

[10] M. R. A. Rashid, G. Rizzo, M. Torchiano, N. Mihindukulasooriya, O. Corcho, and R. García-Castro, "Completeness and consistency analysis for evolving knowledge bases," *Journal of Web Semantics*, Nov. 2018, doi: 10.1016/j.websem.2018.11.004.

[11] "An introduction to Datalog." https://blogit.michelin.io/an-introduction-to-datalog/ (accessed May 11, 2023).

[12] L. Liu and M. T. Özsu, Eds., *Encyclopedia of database systems*. Boston, MA: Springer US, 2009.

[13] "RDFS vs. Owl." https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/rdfs-vs-owl/ (accessed May 11, 2023).

[14] L. Ho, S. Arch-int, E. Acar, S. Schlobach, and N. Arch-int, "An argumentative approach for handling inconsistency in prioritized Datalog ± ontologies," *AIC*, vol. 35, no. 3, pp. 243–267, Sep. 2022, doi: 10.3233/AIC-220087.

[15] "1 Datalog Module Language." https://docs.racket-lang.org/datalog/datalog.html (accessed May 11, 2023).

[16] T. Ajileye, B. Motik, and I. Horrocks, "Datalog Materialisation in Distributed RDF Stores with Dynamic Data Exchange," in *The semantic web – ISWC 2019: 18th international semantic web conference, auckland, new zealand, october 26–30, 2019, proceedings, part I*, vol. 11778, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon, Eds. Cham: Springer International Publishing, 2019, pp. 21–37.

[17] B. Scholz, H. Jordan, P. Subotić, and T. Westmann, "On fast large-scale program analysis in Datalog," in *Proceedings of the 25th International Conference on Compiler Construction - CC 2016*, New York, New York, USA, Mar. 2016, pp. 196–206, doi: 10.1145/2892208.2892226.

[18] B. T. Loo and W. Zhou, *Declarative Networking*. Cham: Springer International Publishing, 2012.

[19] M. Huth and M. D. Ryan, "Logic in computer science - modelling and reasoning about systems (2. ed.).," Jan. 2004.

[20] G. Antoniou and F. Van Harmelen, "A Semantic Web Primer," May 2008. https://mitpress.mit.edu/9780262012423/a-semantic-web-primer/ (accessed May 13, 2023).

[21] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about Datalog (and never dared to ask)," *IEEE Trans. Knowl. Data Eng.*, vol. 1, no. 1, pp. 146–166, Mar. 1989, doi: 10.1109/69.43410.

[22] "RDF 1.1 Concepts and Abstract Syntax."

https://www.w3.org/TR/rdf11-concepts/ (accessed May 13, 2023).
[23] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data - The Story So Far," *Int. J. Semant. Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, Jul. 2009, doi: 10.4018/jswis.2009081901.
[24] "RDF Model and Syntax." https://www.w3.org/TR/WD-rdf-syntax-971002/ (accessed May 13, 2023).
[25] T. Berners-Lee, R. Fielding, and L. Masinter, *Uniform resource identifier (URI): generic syntax*. RFC Editor, 2005.
[26] "RDF/XML Syntax Specification (Revised)." https://www.w3.org/2001/sw/RDFCore/TR/WD-rdf-syntax-grammar-20030117/ (accessed May 13, 2023).
[27] "Turtle - Terse RDF Triple Language." https://www.w3.org/TeamSubmission/2008/SUBM-turtle-20080114/ (accessed May 13, 2023).
[28] "N-Triples." https://www.w3.org/2001/sw/RDFCore/ntriples/ (accessed May 13, 2023).
[29] "JSON-LD 1.1." https://www.w3.org/TR/json-ld11/ (accessed May 13, 2023).
[30] "SPARQL Query Language for RDF." https://www.w3.org/TR/rdf-sparql-query/ (accessed May 13, 2023).
[31] "SPARQL 1.1 Overview." https://www.w3.org/TR/sparql11-overview/ (accessed May 13, 2023).
[32] "SPARQL 1.1 Federated Query." https://www.w3.org/TR/sparql11-federated-query/ (accessed May 13, 2023).
[33] *Semantic web for the working ontologist*. Elsevier, 2011.
[34] "RDF Schema 1.1." https://www.w3.org/TR/rdf-schema/ (accessed May 13, 2023).
[35] "RDF Primer." https://www.w3.org/TR/rdf-primer/ (accessed May 13, 2023).
[36] G. Gottlob, G. Orsi, A. Pieris, and M. Šimkus, "Datalog and its extensions for semantic web databases," in *Reasoning Web. Semantic Technologies for Advanced Query Answering: 8th International Summer School 2012, Vienna, Austria, September 3-8, 2012. Proceedings*, vol. 7487, T. Eiter and T. Krennwallner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 54–77.
[37] A. Rossi, D. Barbosa, D. Firmani, A. Matinata, and P. Merialdo, "Knowledge graph embedding for link prediction," *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 2, pp. 1–49, Apr. 2021, doi: 10.1145/3424672.
[38] T. Badriyah, E. T. Wijayanto, I. Syarif, and P. Kristalina, "A hybrid recommendation system for E-commerce based on product description and

user profile," in *2017 Seventh International Conference on Innovative Computing Technology (INTECH)*, Aug. 2017, pp. 95–100, doi: 10.1109/INTECH.2017.8102435.

[39] H. Shakibian and N. M. Charkari, "Statistical similarity measures for link prediction in heterogeneous complex networks," *Phys. A: Stat. Mech. Appl*, vol. 501, pp. 248–263, Jul. 2018, doi: 10.1016/j.physa.2018.02.189.

[40] C. Wang, V. Satuluri, and S. Parthasarathy, "Local probabilistic models for link prediction," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Oct. 2007, pp. 322–331, doi: 10.1109/ICDM.2007.108.

[41] T. Zhou, L. Lü, and Y.-C. Zhang, "Predicting missing links via local information," *Eur. Phys. J. B*, vol. 71, no. 4, pp. 623–630, Oct. 2009, doi: 10.1140/epjb/e2009-00335-8.

[42] D. Liben-Nowell and J. Kleinberg, "The link prediction problem for social networks," in *Proceedings of the twelfth international conference on Information and knowledge management - CIKM '03*, New York, New York, USA, Nov. 2003, p. 556, doi: 10.1145/956863.956972.

[43] M. Fire, L. Tenenboim, O. Lesser, R. Puzis, L. Rokach, and Y. Elovici, "Link prediction in social networks using computationally efficient topological features," in *2011 IEEE Third Int'l Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third Int'l Conference on Social Computing*, Oct. 2011, pp. 73–80, doi: 10.1109/PASSAT/SocialCom.2011.20.

[44] M. Zhang and Y. Chen, "[1802.09691] Link Prediction Based on Graph Neural Networks," *arXiv*, Feb. 2018.

[45] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, New York, New York, USA, Aug. 2016, pp. 1105–1114, doi: 10.1145/2939672.2939751.

[46] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv*, 2016, doi: 10.48550/arxiv.1609.02907.

[47] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *arXiv*, 2017, doi: 10.48550/arxiv.1710.10903.

[48] M. Zhang and Y. Chen, "Link Prediction Based on Graph Neural Networks," *arXiv*, 2018, doi: 10.48550/arxiv.1802.09691.

[49] J. de Bruijn, E. Franconi, and S. Tessaris, "Logical reconstruction of RDF and ontology languages," in *Principles and practice of semantic web reasoning*, vol. 3703, F. Fages and S. Soliman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 65–71.

[50] "Bassem-Makni/NMT4RDFS: Neural Machine Translation for RDFS reasoning: code and datasets for 'Deep learning for noise-tolerant RDFS reasoning' http://www.semantic-web-journal.net/content/deep-learning-noise-tolerant-rdfs-reasoning-4." https://github.com/Bassem-Makni/NMT4RDFS/ (accessed May 15, 2023).

[51] M. Zneika, D. Vodislav, and D. Kotzinos, "Quality metrics for RDF graph summarization," *SW*, vol. 10, no. 3, pp. 555–584, Apr. 2019, doi: 10.3233/SW-190346.

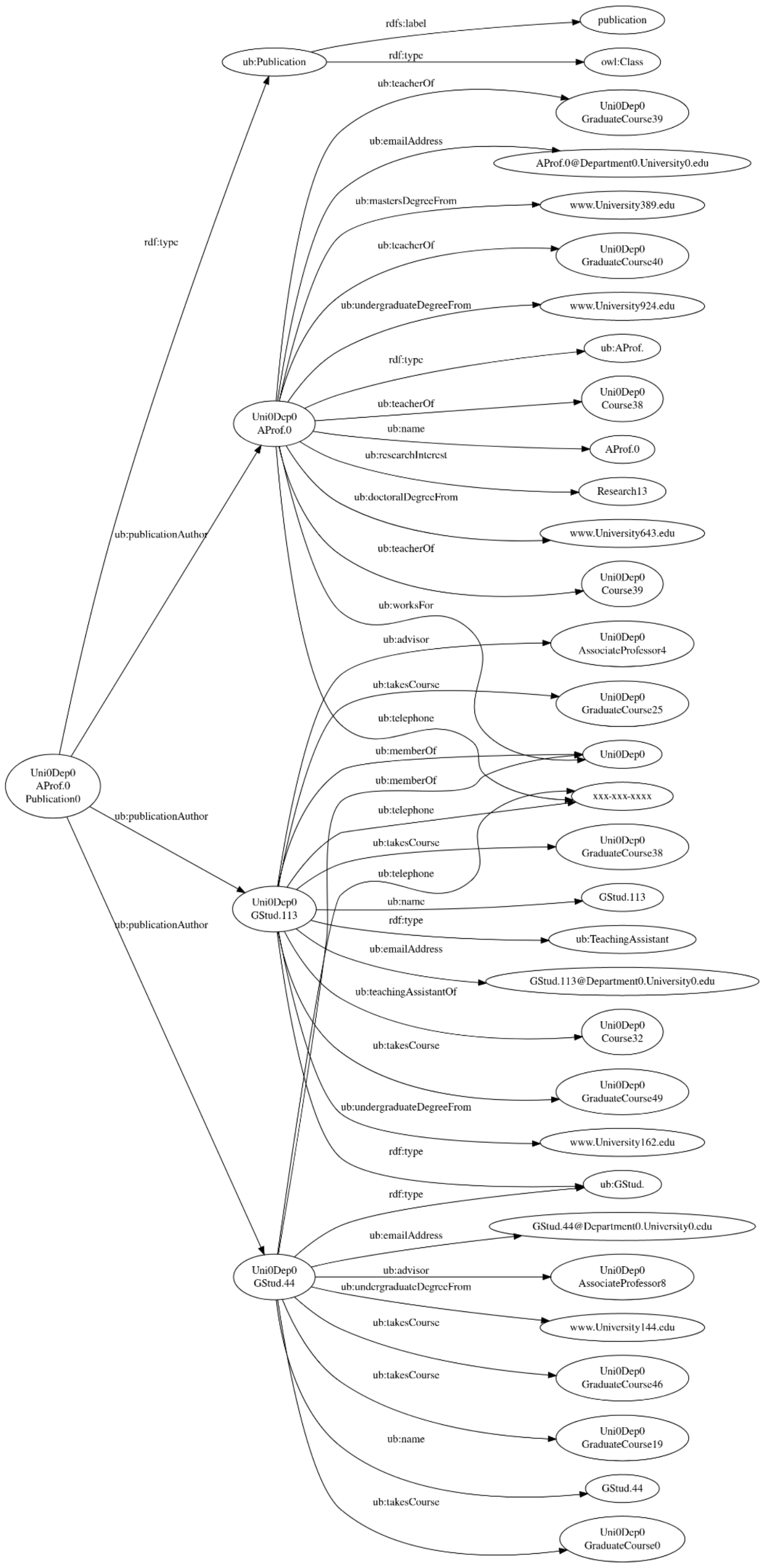
Non-exclusive license to reproduce the thesis and make the thesis public

I, Mart Traagel

1. grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis Comparative Analysis of Deterministic and Graph Neural Network Based RDFS Materialization Methods supervised by Bruno Rucy Carneiro Alves de Lima
2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in points 1 and 2.
4. I confirm that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Mart Traagel
17/05/2023

Appendix A. Fully materialized “Publication0” graph.



ub:Publication
rdfs:label
publication
rdf:type
owl:Class
ub:teacherOf
Uni0Dep0 GraduateCourse39
ub:emailAddress
AProf.0@Department0.University0.edu
ub:mastersDegreeFrom
www.University389.edu
ub:teacherOf
Uni0Dep0 GraduateCourse40
ub:undergraduateDegreeFrom
www.University924.edu
rdf:type
ub:AProf.
ub:teacherOf
Uni0Dep0 Course38
ub:name
AProf.0
ub:researchInterest
Research13
ub:doctoralDegreeFrom
www.University643.edu
ub:teacherOf
Uni0Dep0 Course39
ub:worksFor
Uni0Dep0 AProf.0
rdf:type
ub:publicationAuthor
Uni0Dep0 AProf.0 Publication0
ub:advisor
Uni0Dep0 AssociateProfessor4
ub:takesCourse
Uni0Dep0 GraduateCourse25
ub:telephone
ub:memberOf
Uni0Dep0
ub:memberOf
xxx-xxx-xxxx
ub:telephone
ub:takesCourse
Uni0Dep0 GraduateCourse38
ub:telephone
ub:publicationAuthor
Uni0Dep0 GStud.113
ub:name
GStud.113
rdf:type
ub:TeachingAssistant
ub:emailAddress
GStud.113@Department0.University0.edu
ub:teachingAssistantOf
Uni0Dep0 Course32
ub:publicationAuthor
ub:takesCourse
Uni0Dep0 GraduateCourse49
ub:undergraduateDegreeFrom
www.University162.edu
rdf:type
ub:GStud.
rdf:type
GStud.44@Department0.University0.edu
ub:emailAddress
Uni0Dep0 GStud.44
ub:advisor
Uni0Dep0 AssociateProfessor8
ub:undergraduateDegreeFrom
www.University144.edu
ub:takesCourse
Uni0Dep0 GraduateCourse46
ub:takesCourse
Uni0Dep0 GraduateCourse19
ub:name
GStud.44
ub:takesCourse
Uni0Dep0 GraduateCourse0