3-2-2021

Scalable, Efficient and Precise Natural Language Processing in the Semantic Web

Shane Michael Peelar
University of Windsor

Scalable, Efficient and Precise Natural Language Processing in the Semantic Web

by

Shane Peelar

A Dissertation
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada

2020

Scalable, Efficient and Precise Natural Language Processing in the Semantic Web

by

Shane Peelar

APPROVED BY:

D. Z. Inkpen, External Examiner
University of Ottawa

R. J. Caron
Department of Mathematics and Statistics

J. Lu
School of Computer Science

P. M. Zadeh
School of Computer Science

R. A. Frost, Advisor
School of Computer Science

December 17, 2020

Declaration of Co-Authorship / Previous Publication

I. Co-Authorship

I hereby declare that this thesis incorporates material that is result of joint research undertaken under the supervision of Dr. Richard A. Frost. The collaboration is covered in Chapters 3 through 7 of the dissertation. In all cases, the key ideas, primary contributions and writing were performed by Shane Peelar (the candidate) and Dr. Richard A. Frost (the supervisor) as primary authors and contributors.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Previous Publication

This thesis includes 5 original papers that have been previously published/-submitted for publication in peer reviewed journals, as follows:

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

Thesis Chapter	Publication title/full citation	Publication Status
Chapter 3	An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore	Published
Chapter 4	A New Data Structure for Processing Natural Language Database Queries	Published
Chapter 5	A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore	Published
Chapter 6	A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores	Published
Chapter 7	Accommodating Negation in an Efficient Event-Based Natural Language Query Interface to the Semantic Web	Published

III. General

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

The Internet of Things (IoT) is an emerging phenomenon in the public space. Users with accessibility needs could especially benefit from these "smart" devices if they were able to interact with them through speech. This thesis presents a Compositional Semantics and framework for developing extensible and expressive Natural Language Query Interfaces to the Semantic Web, addressing privacy and auditability needs in the process. This could be particularly useful in healthcare or legal applications, where confidentiality of information is a key concern.

Dedication

This Thesis is dedicated to my Mémé and Pépé, Theresa and Alfred Bombardier

Acknowledgements

I'd like to thank:

Dr. Richard A. Frost, whom I've been working with since my first semester here at the University of Windsor, for his guidance in my work and the many opportunities he has provided me over the years to participate in research. You were the first professor I had a chance to have a conversation with when I started my Undergraduate degree here in 2009, and ultimately it is our conversations during the labs of 60-100 that got me seriously interested in functional programming in the first place.

My external examiner Dr. Diana Zaiu Inkpen, my external reader Dr. Richard Caron, my internal readers Dr. Jianguo Lu and Dr. Pooya Moradian Zadeh for their constructive feedback on my dissertation and the defence presentation.

Dr. Robert D. Kent for being an incredible human being and a great friend. I simply can't put in words how much I enjoy our long discussions about life and just about everything in it.

Bryan St. Amour and Paul Preney for being great friends and colleagues. Our conversations are always so interesting and I'm glad that our paths keep crossing despite us all working in different areas now. I hope they continue to do so. I will always look back fondly on our impromptu meetings with Bob.

My wife, Taylor, for being my person and being incredibly supportive of me both in life and in my work. It's crazy to think about all that's happened in the last 5 years. I'm so grateful to have had you by my side through all of it. Susan, Connor, and Charlotte – thank you for all of your support and letting me into your family, I can't tell you how lucky I am to be a part of it.

My Mom and Dad for all of their support in my life and schooling. It's been a wild ride with a lot of ups and downs, but I think I'm finally here, where I need to be. I hope you can look at what I've accomplished and be proud.

All of the secretaries in the School of Computer Science for their help during my time at the University of Windsor. Whether it was getting me through scheduling nightmares or helping me get things done by the deadlines, your help has been incredibly invaluable to me and I sincerely thank you for your tireless efforts to help us students.

Contents

List of Tables

List of Figures

List of Appendices

List of Abbreviations

Chapter 1

Preface

My involvement in the SpeechWeb project began in 2009 when I had begun my Undergraduate degree in Computer Science at the University of Windsor. Back then, I worked with Dr. Frost as part of my Outstanding Scholars scholarship and became familiar with developing SpeechWeb applications in the Haskell programming language.

Dr. Frost and Dr. Rahmatullah Hafiz had successfully shown in 2012 that it was possible to parse highly ambiguous left-recursive context-free grammars using functional combinators as part of Dr. Hafiz's doctoral dissertation. This opened the doors to a larger NSERC project that was intended to demonstrate that Compositional Semantics is an appropriate choice for building a Natural Language Query Interface to the Semantic Web. At the same time, I set out to work on my Undergraduate thesis, and in 2013, it was presented at ASONAM'13 [13].

In 2014, although I wasn't formally credited, I helped contribute to a paper that was published at the ESWC [12]. I developed the query program that Dr. Frost used in his demonstration at the conference. The main reason I wasn't credited was that I joined the research group after the paper itself had been submitted.

In 2016, I completed my Master's degree with Dr. Frost as my supervisor. My task was to extend the English coverage of the NLQI with chained prepositional phrases. In doing so, I discovered that the word "by" as in "discovered by" could be treated in the same way as a preposition, simplifying the grammar and the semantics

while also allowing for more flexible queries.

In 2017, I began my Doctoral studies aiming at bringing the SpeechWeb to the Semantic Web. This would require both improving the time complexity of query evaluation within the semantics, and also enhancing the query interface with an even broader coverage of English. It would also require allowing the NLQI to run on a wide variety of devices, including those with low power requirements and using the semantics with non-event based triplestores. During this year, Eric Matthews, a fellow graduate student at the University of Windsor, completed his Master's degree thanked me in his Thesis Report for helping him write his thesis [10].

I explored multiple avenues towards solving the problem. I learned about heterogeneous computing and became involved with the Khronos Group OpenCL Working Group with my colleague Paul Preney at the University of Windsor. We published a poster paper at IWOCL 2017 that dealt with extending build systems for heterogeneous OpenCL applications. We were both credited in the OpenCL 2.2 specification released that year [8].

Pursuing heterogeneous computing, at the end of 2017, I obtained an OCE TalentEdge Academic Internship to develop a simulator for Additive Manufacturing processes. I published a paper on leveraging heterogeneous computing for accelerating simulations for these applications in 2018 at CAD Conference in Paris [9]. There, I was invited to submit a full paper to the CAD and Applications Journal which was subsequently published in 2019 [5]. The work is featured in the software APlus which is used by companies around the world for Additive Manufacturing applications.

Although the mathematics behind developing 3D Printing simulations and Natural Language Processing applications are dissimilar, both benefit from the same High Performance Computing techniques for implementing and accelerating them. In particular, I learned a lot about making programs efficient at the microarchitectural level using techniques such as SIMD vectorization and using efficient data layouts. Although none of the papers in this dissertation use heterogeneous computing, notes are made in the Future Work sections of each paper presented where

specialized hardware could be used to accelerate certain computations required by the semantics. In particular, FPGAs, a form of reprogrammable hardware, could be used to accelerate the construction of the FDBR, a datastructure central to the thesis.

Using the microarchitectural-level insight I had gained from this research track, I was able to successfully get the Haskell demo from my Master's thesis running on a low power consumer network router and I realized that the work would be appropriate for Internet of Things (IoT) applications. As we pursued this avenue in the course of my studies, Dr. Frost and I published in both conferences and book series. My contributions to those papers are described as follows:

In 2018, Dr. Frost and I published our first paper in this line of research together at NLIWoD [7], a satellite event of the ISWC, seeking to re-awaken interest in Compositional Semantics as an approach for creating Natural Language Query Interfaces. We also discussed how certain superlatives and graded quantifiers could be handled within the semantics. My contributions to this paper included the implementation of the demonstration program, including the website, as well as the discussion of the FDBR, a fundamental datastructure used in this thesis, and how prepositional phrases are handled.

In 2019, we presented at WEBIST [4], where our paper was nominated for the Best Student Paper Award. Our paper dealt with how our semantics can accommodate "non-compositional" features of English such as superlatives. In particular, we give mention to the $n^2 - n$ binary relations that can be obtained from n-ary events such as those that underlie n-ary transitive verbs. We were subsequently invited to submit an extended paper as a chapter for the WEBIST Springer Book. My contribution to this paper included the discussion of how our approach could be used with traditional relational databases, the demonstration program and implementation, and some of the examples and discussion around them.

In February 2020, we presented at IEEE ICSC [1]. In our IEEE ICSC 2020 paper, we described how to handle transitive verbs with n-ary relations in the semantics and gave a full treatment of the semantics in the Lambda Calculus. We also had

the beginnings of an idea how to adapt the semantics to relational databases. My contributions to this paper include the discussion of how to generate FDBRs from n-ary relations, the formal denotations for chained prepositional phrases, as well as parts of the denotation used for transitive verbs and the examples shown. We were invited to submit to the ASTESJ journal for a special issue, but we did not submit a paper.

In March 2020, we submitted our WEBIST Springer Book [6] paper dealing with improving the computational complexity of our approach by showing how a Compositional Semantics can be memoized. A complete application architecture is presented that permits both online and offline computation of the FDBR datastructures that are fundamental to the semantics. We also show how superlative phrases such as "the most" can be accommodated. My contributions to this paper were all aspects related to memoization, implementation, demonstration, discussion of syntactic and semantic ambiguity, description of the application architecture, denotation for the superlative phrase "the most" including nesting superlatives phrases within chained prepositional phrases, discussion regarding accommodating negation, and the discussion for how to use the approach with relational databases. As of November 2020, our paper is now available via Springer.

In November 2020, we presented our latest paper at WEBIST [2]. Our WEBIST 2020 paper describes how to accommodate negation in our semantics for applications where the Closed World Assumption holds. Denotations for words entailing negation, such as "not", "non" and "no", are presented along with a denotation for transitive verbs that can handle negated expressions. Notably, the denotations for "no", "non" and "not" can be omitted to restore the Open World Assumption where appropriate. My contributions to this paper include the modifications to the semantics to accommodate negation, including integrating negation with the architecture presented in the WEBIST Springer Book chapter, the denotation for "not", and the example queries given.

As of October 2020, we have successfully embedded the semantics directly within the web browser to remove the need of intermediary servers to process queries. The

goal of this is to allow your web browser to communicate directly with Semantic Web triplestores as though they were ordinary websites served over HTTP. Currently, a version of the query interface is available that does this with event-based triplestores. We aim to publish our approach in a functional programming conference as we believe it to be a useful method to create Natural Language Query Interfaces directly in the web browser.

We have also gained industry interest in crossing the gap to non event-based triplestores using Machine Learning approaches [3]. Our next goal is to combine the techniques developed above to query DBPedia with our semantics by leveraging a Machine Learning approach to perform reification on the non-event based triples. This will allow us to directly benchmark our approach against other query interfaces.

As of December 2020, my Master's thesis [11] has been cited 4 times according to Google Scholar and my ASONAM'13 paper has been cited 8 times according to Microsoft Academic.

Bibliography

[1] S. M. Peelar and R. A. Frost. "A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore". In: *IEEE 14th International Conference on Semantic Computing, ICSC 2020, San Diego, CA, USA, February 3-5, 2020*. IEEE, 2020, pp. 257–262. URL: https://doi.org/10.1109/ICSC.2020.00054 (cit. on p. 3).

[2] S. M. Peelar and R. A. Frost. "Accommodating Negation in an Efficient Event-based Natural Language Query Interface to the Semantic Web". In: *Proceedings of the 16th International Conference on Web Information Systems and Technologies, WEBIST 2020, Budapest, Hungary, November 3-5, 2020*. Ed. by M. Marchiori, F. J. D. Mayo, and J. Filipe. SCITEPRESS, 2020, pp. 83–92. URL: https://doi.org/10.5220/0010148500830092 (cit. on p. 4).

[3] Timbr. "Timbr.ai". In: *https://wiki.dbpedia.org/timbr* (2020) (cit. on p. 5).

[4] R. A. Frost and S. M. Peelar. "A New Data Structure for Processing Natural Language Database Queries". In: *Proceedings of the 15th International Conference on Web Information Systems and Technologies, WEBIST 2019, Vienna, Austria, September 18-20, 2019*. 2019, pp. 80–87. URL: https://doi.org/10.5220/0008124300800087 (cit. on p. 3).

[5] S. Peelar, J. Urbanic, R. Hedrick, and L. Rueda. "Real-Time Visualization Of Bead Based Additive Manufacturing Toolpaths using Implicit Boundary Representations". In: *Computer-Aided Design and Applications* 16.5 (2019-01), pp. 904–922. URL: https://doi.org/10.14733%2Fcadaps.2019.904-922 (cit. on p. 2).

[6] S. M. Peelar and R. A. Frost. "A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores". In: *Web Information Systems and Technologies - 15th International Conference, WEBIST 2019, Vienna, Austria, September 18-20, 2019, Revised Selected Papers*. Ed. by A. Bozzon, F. J. D. Mayo, and J. Filipe. Vol. 399. Lecture Notes in Business Information Processing. Springer, 2019, pp. 168–194. URL: https://doi.org/10.1007/978-3-030-61750-9%5C_8 (cit. on p. 4).

[7] R. A. Frost and S. M. Peelar. "An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore". In: *Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, October 8th - 9th, 2018*. Ed. by K.-S. Choi, L. E. Anke, T. Declerck, D. Gromann, J.-D. Kim, A.-C. N. Ngomo, M. Saleem, and R. Usbeck. Vol. 2241. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 1–16. URL: http://ceur-ws.org/Vol-2241/paper-01.pdf (cit. on p. 3).

[8] K. O. W. Group et al. “The OpenCL 2.2 specification, url: https://www. khronos. org/registry/OpenCL/spec s/2.2/pdf”. In: *OpenCL_ API. pdf* (2018) (cit. on p. 2).

[9] S. Peelar, J. Urbanic, R. Hedrick, and L. Rueda. “Real-Time Visualization of Bead Based Additive Manufacturing Toolpaths Using Implicit Boundary Representations”. In: *Proceedings of CAD’18*. CAD, 2018-07. URL: `https://doi.org/10.14733%2Fcadconfp.2018.327-331` (cit. on p. 2).

[10] E. Matthews. “Passive RFID Rotation Dimension Reduction via Aggregation”. In: (2017) (cit. on p. 2).

[11] S. Peelar. “Accommodating prepositional phrases in a highly modular natural language query interface to semantic web triplestores using a novel event-based denotational semantics for English and a set of functional parser combinators”. MA thesis. University of Windsor (Canada), 2016 (cit. on p. 5).

[12] R. A. Frost, J. Donais, E. Matthews, and W. Agboola. “A Demonstration of a Natural Language Query Interface to an Event-Based Semantic Web Triplestore”. In: *ESWC*. Springer LNCS Volume 8798. 2014, pp. 343–348 (cit. on p. 1).

[13] J. A. Donais, R. A. Frost, S. M. Peelar, and R. A. Roddy. “A system for the automated author attribution of text and instant messages”. In: *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM. 2013, pp. 1484–1485 (cit. on p. 1).

Chapter 2

Introduction

The Internet of Things (IoT) is an emerging phenomenon in the public space. Remotely controlled lights, voice assistants connected directly to powerful search engines, and refrigerators that can predict when you'll run out of food are just some examples of where IoT is entering the lives of people around the world. Users with accessibility needs could especially benefit from these "smart" devices if they were able to interact with them through speech.

Privacy is a particular concern about popular voice assistants that are currently in use. They function by sending their queries directly to a remote server for processing which then return results to the user. This architecture makes their use in confidential environments problematic, such as in a medical or law office.

Another concern is about the trustworthiness of the returned results. In expert systems, it's not enough to just have the answer to a query – users need to be confident that the result is indeed correct.

This thesis describes a framework for building NLQIs using Compositional Semantics that addresses these concerns. In particular, the use of a Compositional Semantics guarantees that the results returned will be as correct as the data in the database. It is also auditable: information is available for the returned results that justifies their inclusion in the result. This can be used to verify that the retrieved information indeed exists within the triplestore or database it was retrieved from.

The NLQIs produced with this framework are able to be run directly on the user's

own devices, such as their own computer, smartphone, or even internet router. The actual queries the user makes never leave the device – the only time information leaves the device is when a query to a Semantic Web triplestore needs to be made to satisfy it. For domain-specific applications where confidentiality is important, the server can be maintained entirely within that environment under the user's control.

Our approach relies on a memoized event-based Compositional Semantics (CS) supporting complex linguistic structures such as prepositional phrases, superlatives, and negation. It is *scalable*, in that the architecture can scale from both small to very large triplestores. It is *efficient*, in that it can run on low power hardware. It is *expressive*, capable of handling queries with many complex linguistic features, and finally it is *precise*, as it is based on a Compositional Semantics, where the answers returned are as correct as the information retrieved from the triplestore.

2.1 Thesis Statement

This dissertation contains five papers that address the problem of creating Natural Language Query Interfaces to the Semantic Web. It proves the following statement:

"A scalable, efficient, expressive and precise method for processing natural-language queries to the Semantic Web can be built using a Compositional Semantics."

This dissertation describes research that proves the thesis. The research was originally published in refereed conference papers and book chapters that are reproduced in Chapters 3 through 7. The rest of this chapter introduces the papers above along with their novel contributions and related background information.

Please see Appendix B for a list of all 10 papers related to this Thesis that I authored or co-authored.

2.2 Proof of Concept

Rather than prove the thesis statement mathematically, I have chosen to build an interface that proves that the thesis is correct. It can be found at the following

URL:

`https://speechweb2.cs.uwindsor.ca/solarman/demo_sparql.html`

Additionally, a version that runs entirely within the web browser is available at the following URL:

`https://speechweb2.cs.uwindsor.ca/solarman-wasm/`

A list of example queries is provided on both websites and in the papers presented in this report.

2.2.1 Speech support

Both interfaces are speech enabled – simply click on the microphone icon to the left of the query box and speak a question into your microphone. You may need to give Solarman permission to access your microphone – if this is the case, click "Allow" on the prompt that appears. After speaking, the system will automatically perform the query and use synthesized speech to read the result aloud back to you. For example, try pressing the microphone button and speaking "`who discovered a moon that orbits mars`" into your microphone – you should hear back the answer "`hall`" from the NLQI. Supporting web browsers include Mozilla Firefox and Google Chrome-based browsers at this time, including Microsoft Edge. Internet Explorer is not supported.

2.3 Novel Contributions

We have made several novel contributions while researching the Thesis Statement. They are outlined below:

- A denotation for n-ary transitive verbs in a CS
- A denotation for superlatives and comparatives such as "most" and "the most"

- How the FDBR datastructure can be used to accommodate queries with "non-compositional" features in a CS for a NLQI
- A memoized query evaluation framework for efficiently evaluating NL queries to a triplestore using a CS
- A mapping between event-based triplestores and relational databases, and how our event semantics can handle both types of databases

These contributions are described in more detail in the following chapters of this dissertation.

2.4 Limitations

Our approach is currently geared towards expert systems and domain-specific applications. Although it maintains a very wide coverage of English in queries, it is intended to be used in curated knowledge bases where there is a high degree of certainty about the correctness of the contained information. One property of our approach is that the answer is as correct as what is contained within the databases themselves. Our approach has not yet been formally evaluated against other NLQIs – we provide only qualitative comparisons with other NLQIs in this dissertation. In our approach, a URI identifies an entity or an event uniquely within the universe of discourse. In this thesis we have used URIs which identify entities uniquely within this Windsor project.

Currently, numerical quantifiers such as "one", "two" depend upon the Single Role Assumption, where an event may have at most one entity fulfilling a particular role.

Note that while our demonstration uses our "Solarman" knowledge base to answer NL queries about the solar system, the semantics as presented are highly general and could be adapted to many different types of knowledge bases. One would need to understand the domain-specific aspects of those knowledge bases and provide a vocabulary to facilitate this. For example, with respect to a medical knowledge base, one could answer the query "Does X contraindicate Y?", where X and Y are

names of drugs, by using our denotation for transitive verbs as the denotation of "contraindicate" and selecting the relation underlying that verb as appropriate. This applies to all other syntactic categories as well.

2.5 An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore

Chapter 3 contains a paper that describes the Function Defined by a Binary Relation (FDBR) and how it can be used to answer Natural Language queries to event-based Semantic Web Triplestores. It was presented at NLIWOD 2018, a satellite event of the ISWC.

Briefly, Semantic Web Triplestores are databases that contain *Resource Description Framework (RDF) triples* that describe facts about *entities*. Each component of a triple is a Uniform Resource Identifier (URI) that uniquely identifies an entity within that triplestore. As an example for how these are used, consider the following:

<hall> <discover> <phobos> .

<hall> <discover> <deimos> .

The triples above can be read as "*subject-predicate-object*".

Triplestores are accessible via a query method using an endpoint. The most common query method in use for Semantic Web triplestores is *SPARQL* [3], however, increasingly users are turning to other query methods as well, such as Linked Data Fragments [2] being the recommended choice for querying DBPedia [4].

One problem with entity-based triples such as the example above is that it is difficult to add contextual information to a set of triples. In the example above, it is not clear if Asaph Hall discovered both Phobos and Deimos at the same time or if these were separate events, and it is not clear what year those discoveries took place. The triples must be *reified* [5] to obtain the contextual information.

One method for reification is to use *event-based triplestores*, where the *subject* of each triple identifies an *event* rather than an entity. The example above could be expressed as:

<event1045> <type> <discovery> .

<event1045> <subject> <hall> .

<event1045> <object> <phobos> .

<event1045> <year> <1877> .

<event1046> <type> <discovery> .

<event1046> <subject> <hall> .

<event1046> <object> <deimos> .

<event1046> <year> <1877> .

Event-based triplestores allow additional contextual information about an event to be expressed expressed by simply adding additional triples with that event as the subject. In the example above, it is clear that both discovery events for Deimos and Phobos took place within the same year.

We present a Natural Language Query Interface to event-based triplestores using the Function Defined by a Binary Relation (FDBR), a datastructure first used by Frost in [6] to provide a denotation for binary transitive verbs in a set-theoretic version of Montague Semantics [7]. In 2016, Peelar showed that the FDBR can be used to accommodate chained prepositional phrases in a Compositional Semantics [1]. In this paper, we discuss how the FDBR can be used to answer other types of Natural Language queries as well, including those with superlatives and graded quantifiers.

2.6 A New Data Structure for Processing Natural Language Database Queries

Chapter 4 contains a paper that describes how the FDBR is used to create denotations of other linguistic constructs such as superlatives. It also discusses how the

semantics can be readily adapted to relational databases. This paper was published at WEBIST 2019. We were invited to submit an extended paper to the WEBIST Springer Book, and at the conference we were nominated for Best Student Paper Award.

2.7 A Compositional Semantics for a Wide-coverage Natural-Language Query Interface to a Semantic Web Triplestore

Chapter 5 contains a paper that describes the $n^2 - n$ functions defined by an n-ary relation and how these can be used to accommodate transitive verbs in the semantics with n-ary relations. This paper was presented at IEEE ICSC 2020. We were invited to submit to the ASTESJ journal for a special issue, but we did not submit a paper.

2.8 A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores

Chapter 6 contains a paper that describes how to adapt our semantics to relational databases. It provides a full treatment of the application architecture of our query interface, and discusses strategies for handling both syntactic and semantic ambiguity in the returned results over both speech and text modalities. It also describes how the semantics are memoized to improve their computational complexity and discusses that framework enables FDBRs to be precomputed and cached offline for fast retrieval.

One of the key ideas behind this paper is to memoize the semantics by using the expression tree obtained from the query itself. Each expression is uniquely named

and the results are cached from the triplestore, drastically reducing the number of evaluations that need to be performed during a query. The same architecture can be used to cache and generate FDBRs offline for fast retrieval later. It addresses both the efficiency and scalability aspects of the thesis statement.

We were invited to submit this paper for inclusion in the WEBIST Springer Book, where it has been published.

2.9 Accommodating Negation in an Efficient Event-Based Natural Language Query Interface to the Semantic Web

Chapter 7 contains a paper that describes how to accommodate negation within the semantics. This conference paper was presented at WEBIST 2020. The key idea behind this paper is to track cardinality throughout the semantics by introducing a new triplestore querying primitive to obtain the cardinality of the triplestore. Negation in general only holds if the *Closed World Assumption* can be satisfied. Informally, the Closed World Assumption can be characterized by the statement:

"*The absence of evidence can be assumed as being evidence of absence*".

For example, if a particular entity p is not explicitly stated as being a member of the "person" set, then it can be assumed that p is not a member of that set. The *Open World Assumption* on the other hand does not assume this statement to be true. In the previous example, this would mean that p cannot be assumed as not being a member of the "person" set unless there is an explicit statement of non-membership elsewhere in the database.

RDF itself is built on the Open World Assumption, however certain domain-specific triplestores may have enough information such that the Closed World Assumption is valid for that triplestore. Where it is not, the denotations for "not", "non", and "no" can be omitted, restoring the Open World Assumption that underlies the Semantic Web. This flexibility allows our approach to be used in expert

systems and domain-specific applications where either assumption is appropriate.

2.10 Notation

In addition to standard Lambda Calculus and set-theory notation, the following notation is used in this dissertation:

- "`<subject> <predicate> <object> .`" denotes an RDF triple
- $\mathrm{e_x}$ denotes the entity x
- ev_{1234} denotes the event #1234
- $\|x\|$ is the mathematical denotation of the phrase x
- x_{set} is the set of all entities that are members of x
- x_{pred} the logical predicate associated with the word x
- x_{FDBR} is the FDBR of all entities that are members of x (a datastructure first introduced in Chapter 3)
- Queries are written in a `monospaced` font
- *property* is used to denote a property, role, or type of an event.
- *name* is used to indicate query results and objects in the real world.

For example, $\mathrm{e}_{\mathrm{phobos}}$ is the mathematical object representing *phobos*, a moon that orbits Mars.

Bibliography

[1] S. Peelar. "Accommodating prepositional phrases in a highly modular natural language query interface to semantic web triplestores using a novel event-based denotational semantics for English and a set of functional parser combinators". MA thesis. University of Windsor (Canada), 2016 (cit. on p. 13).

[2] R. Verborgh, M. Vander Sande, P. Colpaert, S. Coppens, E. Mannens, and R. Van de Walle. “Web-Scale Querying through Linked Data Fragments.” In: *LDOW*. Citeseer. 2014 (cit. on p. 12).

[3] S. Harris, A. Seaborne, and E. Prud’hommeaux. “SPARQL 1.1 query language”. In: *W3C recommendation* 21.10 (2013) (cit. on p. 12).

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. “DBpedia: A Nucleus for a Web of Open Data”. In: *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*. ISWC’07/ASWC’07. Busan, Korea: Springer-Verlag, 2007, pp. 722–735. URL: `http://dl.acm.org/citation.cfm?id=1785162.1785216` (cit. on p. 12).

[5] G. Antoniou and F. Van Harmelen. *A semantic web primer*. MIT press, 2004 (cit. on p. 12).

[6] R. Frost and J. Launchbury. “Constructing natural language interpreters in a lazy functional language”. In: *The Computer Journal* 32.2 (1989), pp. 108–121 (cit. on p. 13).

[7] D. Dowty, R. Wall, and S. Peters. *Introduction to Montague Semantics*. Dordrecht, Boston, Lancaster, Tokyo: D. Reidel Publishing Company, 1981 (cit. on p. 13).

Chapter 3

An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore

This paper was published as:

R. A. Frost and S. M. Peelar. "An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore". In: *Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, October 8th - 9th, 2018*. Ed. by K.-S. Choi, L. E. Anke, T. Declerck, D. Gromann, J.-D. Kim, A.-C. N. Ngomo, M. Saleem, and R. Usbeck. Vol. 2241. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 1–16. URL: `http://ceur-ws.org/Vol-2241/paper-01.pdf`

It was featured at NLIWoD 2018, a satellite event of the ISWC.

3.1 Introduction

One of the major advantages of the semantic web is that data on a topic can be added with little knowledge of the way in which existing data is stored. This is particularly the case with reified semantic web triplestores, where people can add many properties such as the time, location, and implement used, to a triple such as "<hall> <discovered> <phobos>". For example, consider an event-based triplestore containing the following triples, amongst others, where we use bare names for URIs:

<event1030> <type> <discovery> .

<event1030> <subject> <hall> .

<event1030> <object> <phobos> .

Additional data can be added as follows:

<event1030> <date> <1877> .

<event1030> <implement> <refractor_telescope_1> .

<event1030> <location> <us_naval_observatory> .

Ideally, it should be possible to query these and other triples, using an extensible Natural-Language Query Interface (NLQI).

In order to facilitate the extension of an NLQI, it helps if the query language is based on a compositional semantics such as Montague Semantics (MS) [29] or a version of it, and if the language processor is highly modular. Such an NLQI to an online event-based triplestore has been constructed and is available through a web interface, which is discussed in Section 3.2. The NLQI can accommodate common and proper nouns, adjectives, conjunction and disjunction, nested quantifiers, intransitive and transitive n-ary verbs, and chained complex prepositional phrases (PPs). The NLQI is implemented as an executable specification of an attribute grammar (EAG) using parser combinators in the pure functional programming language Haskell. Our parser/interpreter can handle ambiguous left-recursive grammars and fully dependent synthesized and inherited attributes. We begin in section 3.2 with a demonstration of our NLQI. In section 3.3, we discuss our modification

to Montague Semantics and our compositional event semantics. In section 3.4 we discuss quantifier scoping. In section 3.5, we describe our implementation of the NLQI as an EAG. In section 3.6, we discuss how our interface can be extended. In section 3.7, we discuss related work. We conclude in section 3.8.

The motivation for this work is to rekindle an interest in Montague-like Compositional Semantics (CS) for query processing. Compositional Semantics have many benefits which have been exploited in many computing applications, including the facilitation of extensibility.

3.2 A Demonstration of our NLQI

We have built a small "Solarman" triplestore containing approximately 22,000 facts about the moons in our solar system, the planets they orbit, and the people who discovered them, when, where and with which telescope or other device. Our NLQI can answer many questions with respect to the "Solarman" triplestore, but no other questions yet. We have installed our NLQI to Solarman on a server and also on a home wireless router to ensure that our approach requires only minimal computing power (the answer time on the router is as fast as on the server.) The triplestore is stored using the Virtuoso semantic web software which supports a SPARQL Protocol and RDF Query Language (SPARQL) endpoint. Our NLQI is accessible through the following web page, which contains example queries, and lists of words and categories of words that can be used in queries:

`http://speechweb2.cs.uwindsor.ca/solarman4/demo_sparql.html`

Event-based triples consist of three fields: an event identifier, a relationship type, and a type or entity identifier. For example, the facts that Hall discovered Phobos in 1877 using a refractor telescope at the US Naval Observatory can be encoded with the triples above. The extra facts, in addition to the type, subject and object of the event enable evaluation of queries containing PPs.

The triples in our triplestore can be accessed by following the link to our prototype NLQI. Our processor uses basic SPARQL retrieval commands to retrieve sets

of entities and events of a given type, and entities which are the actors/properties of a given event. Our NLQI takes advantage of the optimized retrieval operators in the SPARQL endpoint to our triplestore. The functions defined by sets of events are computed in Haskell as needed during the evaluation of the queries.

The following queries illustrate that quantifier scoping is leftmost-outermost:

`every telescope was used to discover a moon` ⇒ *True*

`a moon was discovered with every telescope` ⇒ *False* (No single moon was discovered using every telescope in our database)

`a telescope was used by hall to discover two moons` ⇒ *True*

`which moons were discovered with two telescopes`
⇒ *halimede laomedeia sao themisto*

`hall discovered a moon with two telescopes` ⇒ *False*

`who discovered deimos with a telescope that was used to discover every moon that orbits mars` ⇒ *hall;hall*

`who discovered a moon with two telescopes`
⇒ *nicholson science_team_18 science_team_2*

`how was sao discovered` ⇒ *blanco_telescope canada-france-hawaii_telescope*

`how many telescopes were used to discover sao` ⇒ *2*

`who discovered sao` ⇒ *science_team_18*

`how did science_team_18 discover sao`
⇒ *blanco_telescope canada-france-hawaii_telescope*

`which planet is orbited by every moon that was discovered by two people` ⇒ *saturn; none* (multiple results are returned as the query is ambiguous)

`which person discovered a moon in 1877 with every telescope that was used to discover phobos` ⇒ *hall; none*

3.3 Our Modification to Montague Semantics

3.3.1 Modifying MS to a "set-based" MS

We use sets of entities rather than characteristic functions (unary-predicates) in order to make the implementation of the semantics computationally tractable:

Original MS [29]:

$$
\begin{aligned}
& \|every\ moon\ spins\| \\
\Longrightarrow\ & (\|every\ moon\|)\ \|spins\| \\
\Longrightarrow\ & (\lambda pq.(\forall x)\ (p\ x \Rightarrow q\ x)\ \text{moon}_{pred})\ \text{spins}_{pred} \\
\Longrightarrow\ & (\lambda q.(\forall x)\ (\text{moon}_{pred}\ x \Rightarrow q\ x))\ \text{spins}_{pred} \\
\Longrightarrow\ & (\forall x)\ \text{moon}_{pred}\ x \Rightarrow \text{spins}_{pred}\ x \\
\Longrightarrow\ & \mathit{True}
\end{aligned}
$$

Requiring moon_{pred} to be applied to all entities in the universe of discourse. In the set-based MS, the denotation of a noun, adjective or intransitive verb is a set of entities of type denoted by *es*. Montague's denotation of the word "every" is modified to:

$$
\begin{aligned}
& \|every\ moon\ spins\| \\
\Longrightarrow\ & (\|every\ moon\|)\ \|spins\| \\
\Longrightarrow\ & (\lambda st.s \subseteq t)\ \text{moon}_{set}\ \text{spin}_{set} \\
\Longrightarrow\ & (\lambda t.\text{moon}_{set} \subseteq t)\ \text{spin}_{set} \\
\Longrightarrow\ & \text{moon}_{set} \subseteq \text{spin}_{set} \\
\Longrightarrow\ & \mathit{True}
\end{aligned}
$$

Which is computationally tractable if the representations of sets of moons and things that spin can be readily retrieved. Proper nouns denote functions of type $es \rightarrow Bool$.

$$\|phobos\| = \lambda s.\text{e}_{\text{phobos}} \in s$$

Then:

$$\begin{aligned} & \|phobos\ spins\| \\ \Longrightarrow & (\lambda s.\mathrm{e}_{\mathrm{phobos}} \in s) \\ \Longrightarrow & \mathrm{e}_{\mathrm{phobos}} \in \mathrm{spin}_{set} \\ \Longrightarrow & \mathit{True} \qquad \text{(if Phobos spins.)} \end{aligned}$$

All denotations are modified from Montague's approach to use sets rather than characteristic functions.

3.3.2 Events

In order to accommodate n-ary verbs ($n > 2$) and PPs, we integrate event semantics with MS using ideas from Davidson et al.[9], Rothstein [19] and Champollion [5]. Our basic idea is to modify the above to return sets of pairs (rather than sets of entities) as intermediate results from evaluating the denotations of phrases. Each pair contains an entity paired with a set of events. In some cases, the set of events can be thought of as justifying why the entity is in the result. For example the result of evaluating the phrase "`discover phobos`" contains the pair $(\mathrm{e}_{\mathrm{hall}}, \{\mathrm{ev}_{1030}\})$.

3.3.3 An Explicit Denotation for Transitive Verbs

MS does not provide an explicit denotation for transitive verbs and deals with them using a syntactic manipulation rule at the end of rewriting the expressions containing them (see page 216 of [29]). The basic idea underlying our approach is to regard each n-ary verb as defining $n^2 - n$ functions between the entities in the n roles in the events associated with that verb.

A Denotation of transitive verbs without events in the semantics

In a simple version of our semantics, 2-place transitive verbs denote functions from a possibly empty list of at most one termphrase to a set of pairs of type (e, es) where e is an entity and es is a set of entities. The function computes the answer

by using data that is retrieved from the datastore as needed. Consider the verb "discover" which we use in our examples. In our triplestore, each discovery event has 5 roles: "*subject*" (agent), "*object*" (theme), "*implement*", "*year*" and "*location*". The triplestore defines 20 binary relations between these 5 roles: *subject* → *object*, *subject* → *implement*, *subject* → *year*, *subject* → *location*, *object* → *subject*, etc. For example, the facts that Hall discovered Phobos and Deimos and Kuiper discovered Miranda and Nereid, are represented as follows:

$$\text{discover}_{rel:subject \to object} = \{(\text{e}_{\text{hall}}, \text{e}_{\text{phobos}}), (\text{e}_{\text{hall}}, \text{e}_{\text{deimos}}), (\text{e}_{\text{kuiper}}, \text{e}_{\text{miranda}}), (\text{e}_{\text{kuiper}}, \text{e}_{\text{nereid}}), etc \ldots\}$$

For every n-ary verb there are $n^2 - n$ binary relations represented by the events associated with that verb. Each binary relation can be converted to a function, by "collecting", into a set, all values in the codomain that are associated with each value in the domain of the relation, and creating a pair consisting of the value from the domain paired with that set. In 2016, Peelar called this induced function the *Function Defined by a Binary Relation (FDBR)* [4]:

$$\text{FDBR}(rel) = \{(x, \text{image}_x) \mid (\exists e)\ (x,e) \in rel\ \ \&\ \ \text{image}_x = \{y \mid (x,y) \in rel\}\}$$

For example, the function defined by the $\text{discover}_{rel:subject \to object}$ relation above is:

$$\text{FDBR}(\text{discover}_{rel:subject \to object}) = \{(\text{e}_{\text{hall}}, \{\text{e}_{\text{phobos}}, \text{e}_{\text{deimos}}\}), (\text{e}_{\text{kuiper}}, \{\text{e}_{\text{miranda}}, \text{e}_{\text{nereid}}\}), etc \ldots\}$$

In such functions, we shall refer to the first value in a pair as the "subject" and the value in the second place as the "set of objects". Consider the query '`who discovered phobos`": the function which is the denotation of "`discovered`" computes $\text{FDBR}(\text{discover}_{rel:subject \to object})$ and then applies the function which is the denotation of "`phobos`" to the set of objects in every pair $(subj, objs)$ which is a mem-

ber of $\mathrm{FDBR}(\mathrm{discover}_{rel:subject \to object})$. For every pair which returns a value of *True*, the subject of the pair is added to the result. The final result of "`discovered phobos`" is a set of pairs, each consisting of every subject which was mapped by $\mathrm{FDBR}(\mathrm{discover}_{rel:subject \to object})$ to a set of objects which contains $\mathrm{e}_{\mathrm{phobos}}$. That is, every entity that discovered phobos paired with the set of events which justify that entity being in the answer. The answer to this example query includes $\mathrm{e}_{\mathrm{hall}}$. Similarly, the query "`who discovered a moon`" is processed analogously to the above, with the denotation of "`a moon`" being applied to the set of objects in every pair in $\mathrm{FDBR}(\mathrm{discover}_{rel:subject \to object})$, and if *True*, the associated subject is added to the result. Every entity that discovered a moon is in the result.

If no termphrase follows the transitive verb, all subjects of pairs that are in $\mathrm{FDBR}(\mathrm{discover}_{rel:subject \to object})$ are returned as the answer. For example, the answer to the query "`who discovered`" is the set of all entities who discovered anything.

Denotation of transitive verbs with events

In order to take advantage of the extra knowledge represented by events, we modify the above so that the denotation of a transitive verb is a function from a list of at most one termphrase and a possibly empty list of PPs to a set of pairs of type (e, evs) where e is an entity and evs is a set of events. The function first computes a discover relation from subjects of discover events to those events:

$$\mathrm{discover}_{rel:subject} = \{(\mathrm{e}_{\mathrm{hall}}, \mathrm{ev}_1), (\mathrm{e}_{\mathrm{hall}}, \mathrm{ev}_2), (\mathrm{e}_{\mathrm{kuiper}}, \mathrm{ev}_3), (\mathrm{e}_{\mathrm{kuiper}}, \mathrm{ev}_4), etc \ldots\}$$

The FDBR of this binary relation is then computed:

$$\mathrm{FDBR}(\mathrm{discover}_{rel:subject}) = \{(\mathrm{e}_{\mathrm{hall}}, \{\mathrm{ev}_1, \mathrm{ev}_2\}), (\mathrm{e}_{\mathrm{kuiper}}, \{\mathrm{ev}_3, \mathrm{ev}_4\}), etc \ldots\}$$

Consider the query "`who discovered phobos`": the function which is the denotation of "`discovered`" computes $\mathrm{discover}_{rel:subject}$ and then applies the function which is the denotation of "`phobos`" to the set of objects of the events in every pair $(subj, evs)$

which is a member of $\textsf{discover}_{rel:subject}$. For every pair which returns *True*, the subject and set of events is added to the resulting denotation. The final resulting denotation of "`discovered phobos`" is a set of pairs consisting of subjects which were mapped by $\textsf{discover}_{rel:subject}$ to a set of events whose objects contains $\mathsf{e}_{\text{phobos}}$. The answer to this example query includes $(\mathsf{e}_{\text{hall}}, \{\mathsf{ev}_{1030}\})$. Similarly, the query "`who discovered a moon`" is processed analogously to the above, with the denotation of "`a moon`" being applied to the set of objects from the set of events in every pair in $\textsf{discover}_{rel:subject}$, and if *True*, the pair is added to the result. If no termphrase or PP follows the transitive verb, all pairs in $\textsf{discover}_{rel:subject}$ are returned as the answer . For example, the answer to the query "`who discovered`" is the set of all entities who discovered anything, paired with the set of events of type discovery in which they were the subject.

Dealing with prepositional phrases

We begin by noting that we treat passive forms of verbs, such "`discovered by hall`" similarly to "`discovered with a telescope`" [4]. Prepositional phrases such as "`with a telescope`" are treated similarly to the method described in Section 3.3.3 except that the termphrase following the preposition is applied to the set of entities that are extracted from the set of events in the FDBR function, according to the role associated with the preposition. The result is a "filtered" FDBR which is further filtered by subsequent PPs. For example, consider the query:

`who discovered in 1948 and 1949 with a telescope` $\Rightarrow$ *kuiper*

The calculation here involves computing $\textsf{discover}_{rel:subject}$, then filtering it with the denotation of "`in 1948 and 1949`", then finally filtering it with the denotation of "`with a telescope`".

Choosing the FDBR to compute

The denotation of a verb, for example "`discover`", needs to know which FDBR to compute before PPs are applied. For example, the query "`what was used to discover two moons`" needs $\textsf{discover}_{rel:implement}$, whereas "`who discovered two`

`moons`" needs $\textsf{discover}_{rel:subject}$. In our approach, the choice is made depending on the context in which the verb appears. The denotation of a transitive verb contains the "active" and "passive" properties to be queried depending on the verb voice, along with the event type that corresponds to the underlying relation. The grammar determines whether a transitive verb is used in the active or passive voice and selects the corresponding property in the denotation to form the domain of the FDBR. In the above examples, when "`used`" is in the active voice, it selects the "*subject*" property, but if it is in the passive voice, it selects the "*implement*" property. In both cases, the "*type*" property of the events that the FDBR is built from is "discover_ev".

3.4 Quantifier Scope

We have integrated a Montague-like [29] compositional semantics with our own version of event semantics. There has been much debate by linguists concerning the viability of integrating compositional and event semantics, particularly with respect to quantifier scope. For example, Champollion [5] argues that analysis of quantifier scope does not pose any special problems in an event semantic framework and presents an implementation of a quantificational event semantics that combines with standard treatments of scope-taking expressions in a well-behaved way. The following examples in subsection 3.4.1, which have been tested with our interface, suggest that our approach returns appropriate results for scope-ambiguous queries. In fact, the answer returned is exactly what is expected if the queries are treated as having leftmost, outermost quantifier scope. Below each query-answer pair, we briefly explain how our system computes the answer.

3.4.1 Example Queries Illustrating Quantifier Scoping

a. `every moon that orbits mars and was discovered with a telescope was discovered by a person` $\Rightarrow$ *True*

The evaluation begins by retrieving all of the "`moon`" entities and then inter-

secting this set with the set returned by evaluating $\|$*orbits mars*$\|$, which is obtained by use of the function f_orbit$_{subject \rightarrow object}$ from subjects to the set of objects which they orbit. The function that is the denotation of "`mars`" is then applied by the function denoted by "`orbit`" to all sets of objects that are in the range of the function f_orbit$_{subject \rightarrow object}$. This returns the set of subjects that orbit Mars. Then, This set is intersected with the set of all moons that were discovered with a telescope (which is computed using the function f_discover$_{object \rightarrow implements}$). The set resulting from this intersection is then passed as the first argument to the denotation of "`every`". The second argument to $\|$*every*$\|$ is the set obtained by evaluating the phrase "`discovered by a person`" which is computed by use of the function f_discover$_{object \rightarrow subject}$ from objects discovered to subjects who discovered them. The function that is the denotation of "`a person`" is applied by the function denoted by "`discover`" to all sets of subjects that are in the range of the function f_discover$_{object \rightarrow subject}$ This returns the set of objects that were discovered by a person. $\|$*every*$\|$ applies the subset operator to the two arguments and returns ***True*** if and only if the set of objects $\|$*moon that orbits mars and was discovered with a telescope*$\|$ is a subset of $\|$*discovered by a person*$\|$. In our triplestore, this is the case.

b. `every moon that orbits mars and was discovered by a person was discovered with a telescope` $\Rightarrow$ *True*

Similar explanation to that for query a.

c. `every moon that orbits Neptune was discovered by a person or a team` $\Rightarrow$ *True*

Scoping does not require the person or the team to be the same for all discoveries of the moons that orbit Neptune. $\|$*discovered by a person or a team*$\|$ returns everything that was discovered by any person or any team. This set is tested by $\|$*every*$\|$ to see if it includes all of the entities returned by $\|$*moon that orbits neptune*$\|$.

d. `a telescope or voyager_2 was used to discover every moon that orbits neptune` $\Rightarrow$ *False*

No single telescope nor Voyager 2 was used to discover every moon that orbits Neptune

e. `every moon that orbits neptune was discovered with a telescope or voyager_2` ⇒ *True*

Voyager 2 or at least one, not necessarily the same, telescope was used to discover each of the moons that orbit Neptune.

f. `every moon that was discovered with a telescope was discovered by hall` ⇒ *False*

Some moons were discovered with a telescope but not discovered by Hall.

g. `every moon that was discovered by hall was discovered with a telescope` ⇒ *True*

Hall used a telescope in all of his discoveries of moons.

Our approach appears to be consistent with the "Scope Domain Principle" described by Landman [27]. That is, all quantificational noun phrases must take scope over the event argument. For example, in our semantics, the answer to the query "`hall discovered every moon`" is computed by checking to see if, for every moon *m*, there exists an event of type discovery, with subject Hall and object *m*. Our approach does not compute the answer to "cumulative" readings of queries such as "`who discovered a moon with two telescopes (used simultaneously)`".

3.4.2 Example Queries Illustrating the Scoping of Chained Prepositional Phrases

The following examples illustrate how queries with chained PPs are answered. It should be noted that Halimede, Laomedeia, Sao and Themisto are the only moons that were discovered using two telescopes separately (see queries a to e) and that Nicholson used two telescopes to discover a total of 4 moons, but did not discover any one moon with two telescopes (see queries g to i).

a. `which moons were discovered with two telescopes`
   ⇒ *halimede laomedeia sao themisto*

b. `who used two telescopes to discover a moon`
   ⇒ *nicholson science_ team_ 18 science_ team_ 2*

c. `who discovered sao` ⇒ *science_ team_ 18*

d. `who discovered themisto` ⇒ *science_ team_ 2*

e. `which moon was discovered by science_team_18 with two telescopes`
   ⇒ *halimede laomedeia sao*

f. `what was used to discover sao`
   ⇒ *blanco_ telescope canada-france-hawaii_ telescope*

g. `what did nicholson discover with two telescopes`
   ⇒ *sinope lysithea carme ananke*

h. `which moon was discovered by nicholson with two telescopes`
   ⇒ *none*

i. `which moon was discovered by nicholson with one telescope`
   ⇒ *ananke carme lysithea sinope*

j. `how was sinope discovered` ⇒ *refractor_ telescope_ 2*

k. `how was carme discovered` ⇒ *hooker_ telescope*

l. `how was ananke discovered` ⇒ *hooker_ telescope*

m. `how was lysithea discovered` ⇒ *hooker_ telescope*

n. `what did nicholson discover with one telescope` ⇒ *nothing*

o. `what did nicholson discover with a telescope`
   ⇒ *sinope lysithea carme ananke*

Note that in the above queries, "`one`" and "`two`" are taken to mean "exactly one" and "exactly two". Since Nicholson used multiple telescopes to discover multiple objects, n) returns *nothing* ("`discover with one telescope`" returns an FDBR of all discoverers that used exactly one telescope in all their discoveries, which excludes Nicholson). On the other hand o) relaxes this restriction, yielding the expected result (see Section 3.8). Note that i) differs from n) because "`was discovered with one telescope`" returns an FDBR of all objects that were discovered each with exactly one telescope.

3.5 Implementation

We built our query processor as an executable attribute grammar using the X-SAIGA Haskell parser- combinator library package. The collect function which converts a binary relation to an FDBR is one of the most compute intensive parts of our implementation of the semantics [1]. However, in Haskell, once a value is computed, it can be made available for future use. We have developed an algorithm to compute FDBR(*rel*) in $O(n \lg n)$ time, where n is the number of pairs in *rel*. Alternatively, the FDBR functions can be computed and stored in a cache when the NLQI is offline. Our implementation is amenable to running on low power devices, enabling it for use with the Internet of Things. A version of our query processor exists that can run on a common consumer network router as a proof of concept for this application. The use of Haskell for the implementation of our NLQI has many advantages, including:

1. Haskell's "lazy" evaluation strategy only computes values when they are required, enabling parser combinator libraries to be built that can handle highly ambiguous left-recursive grammars in polynomial time. The accommodation of left recursive grammars simplifies the integration of semantic and syntactic rules in the EAGs, enabling the query processor to be highly modular and extensible.
2. The higher-order functional capability of Haskell allows the direct definition

of higher-order functions that are the denotations of some English words and phrases. For example: *termand* $s\ t = \lambda v.s\ v\ \&\ t\ v$

3. The ability to partially apply functions of n arguments to 1 to n arguments allows the definition and manipulation of denotation of phrases such as "`every moon`", and "`discover phobos`".

4. The availability of the *hsparql* [17] Haskell package enables a simple interface between our semantic processor and SPARQL endpoints to our triplestores.

3.6 Extensibility

A contribution of this paper is to raise awareness of the importance of extensibility of NLQIs to the semantic web. We use the term "extensibility" in the sense that it is used in Software Engineering, meaning the extent to which the implementation takes future growth into consideration, and a measure of the ability to extend the NLQI and the level of effort required to implement the extension.

3.6.1 Design for extensibility

A number of design decisions facilitate future extension of our NLQI:

1. Our query processor is implemented as a highly-modular executable specification of an attribute grammar (AG). AGs were introduced by Knuth [30] and are widely used to define both the syntax and semantics of programming languages. Each syntax rule has one or more attribute rules associated with it. The attribute rules define how the value of synthesized and inherited attributes of the non-terminal defined by the associated syntax rule are computed from attribute values of the terminals and non-terminals that appear on the right-hand side of the syntax rule. There is a close similarity between AGs and Montague Grammars (MGs), although they were developed independently by a Computer Scientist and a linguist respectively. An executable

attribute grammar (EAG) is an AG whose defined language processor is implemented in a programming language such that the program code for the language processor closely resembles the textbook notation for the AG defining the language to be processed. EAGs are ideally suited for implementation of language interpreters for MGs.

2. Our semantics is based on a highly modular and compositional semantics. The similarity of MGs and AGs suggested to us that it should be comparatively easy to implement a Montague-style natural-language query processor as an executable attribute grammar. Frost and Hafiz [18] therefore began by defining memoized functional combinators, corresponding to "`orelse`" and "`then`" that enable language processors to be built as executable specifications of attribute grammars.

3. The dictionary in the Haskell code to facilitate the addition of new words and categories of words to the query language. Our NLQI Haskell code can be accessed at:

   `http://speechweb2.cs.uwindsor.ca/solarman4/src/`

   The code contains a dictionary consisting of entries such as the following:

   ```
   (``person'', Cnoun, [NOUNCLA_VAL $ get_members ``person''])
   ```

   Which defines the word "person" to be a common noun (`cnoun`) whose meaning is a list of attributes, comprising one attribute of type NOUNCLA_VAL whose value is an FDBR extracted from the triplestore by the `get_members` function which returns an FDBR of all entities that are ***subject***s of events of type "*member*" whose object is "*person*". Our parser combinators include a combinator that creates interpreters for different categories of terminals. For example:

   ```
   cnoun = memoize_terminals_from_dictionary Cnoun
   ```

The combinator `memoize_terminals_from_dictionary` scans the dictionary and creates the interpreter `cnoun` (for the terminal category of common nouns) by "orelsing" all of the basic interpreters that it constructs for words in the first field of every triple in the dictionary that has the "constructor" `Cnoun` in the second field. The list of attributes in the third field is integrated into each basic interpreter constructed. The resulting interpreter for the syntactic category is memoized so that its results can be reused in any subsequent pass over the query string by the same interpreter. The query language can be easily extended with new words and new categories of words by simply adding new entries to the dictionary. Note that only bare names need to be used in the dictionary, as the first part of the URI is added by the combinator that makes the basic interpreter for that word.

4. Our EAG implementation is such that individual parsers can be applied to phrases of English rather than whole queries. This allows us to define new words in terms of existing phrases for which we have defined an interpreter. For example:

```
discoverer = meaning_of nouncla ``person who discovered
                                  something''
```

5. Construction of the interpreter as an EAG accommodates ambiguous and left-recursive grammars greatly facilitates the extension of the query language to include new constructs. When grammars are converted to non-left-recursive form (which is often the case when modular top-down parsers are used), this can complicate the specification of semantic rules. For example, the specification of the syntax and associated semantic rules for converting a bit string to its decimal value is much easier if the grammar chosen is left recursive.

3.6.2 Examples of extending the NLQI

We have given examples, in sub-section 6.1 of how we can add single words such as "`person`" and "`discoverer`" to the query language by simply adding an entry to the in-program dictionary. The query Language can also be easily extended with new language constructs by adding new syntax rules to the EAG together with their associated attribute rules. For example, suppose that we want to be able to ask questions such as "`tell me all that you know about hall discovering a moon that orbits mars`" The phrase "`hall discovering a moon that orbits mars`" could be processed using the interpreter for questions which would return the set of two events where Hall discovered Phobos and Deimos. The meaning of the phrase "`tell me all that you know about`" could be designed so that, for each event, a string could be generated: "`hall discovered phobos with refractor_telescope_1 in 1987 at the us_naval_observatory`" and also any other data that had been added about ev_{1030} Another type of question could be "`who discovered which moons`". The meaning of the word "`which`" could be changed temporarily to that of "`a`" and the question "`who discovered a moon`" answered. The resulting FDBR could be returned from the latter question and then used to generate pairs of people and the list of moons they discovered as answer to the original query.

Adding superlatives and graded quantifiers

The second contribution of this paper is to recognize that each FDBR contains more information than we have taken advantage of so far. For example, in computing the answer to the query "`who discovered every moon`". We consider each pair $(subj, objs)$ in $\mathsf{FDBR}(\mathsf{discover}_{rel:subject \rightarrow object})$ independently and apply the meaning of "`every moon`" to the set *objs* in order to determine if the subj should be in the answer. However, for a question such as "`who discovered the most moons that orbit mars`" the whole of the FDBR needs to be processed so that the result contains the subject with the most events representing that subject

discovering a moon that orbits mars. This requires the addition of "`the most`" to the set of quantifiers and the appropriate modification to the denotation of transitive verbs to take advantage of the information available in the appropriate FDBR. A similar modification could be made to accommodate queries containing the words "`earliest`" "`most recently`", using $\mathsf{FDBR}(\mathsf{discover}_{rel:object \to year})$ etc., and queries such as "`which telescope was used to discover most moons`" using $\mathsf{FDBR}(\mathsf{discover}_{rel:implement \to object})$, etc. Accommodating "`the least`" can be similarly achieved by introducing complements into the semantics [2].

3.7 Related Work

Orakel [20] is a portable NLQI which uses a Montague-like grammar and a lambda-calculus semantics to analyze queries. Our approach is similar to Orakel in this respect. However, in Orakel, queries are translated to an expression of first-order logic enriched with predicates for query and numerical operators. These expressions are translated to SPARQL or F-Logic. Orakel supports negation, limited quantification, and simple prepositional phrases. Portability is achieved by having the lexicon customized by people with limited linguistic expertise. It is claimed that Orakel can accommodate n-ary relations with $n \geq 2$. However, no examples are given of such queries being translated to SPARQL.

YAGO2 [7] is a semantic knowledge base containing reified triples extracted from Wikipedia, WordNet and GeoNames, representing nearly 0.5 billion facts. Reification is achieved by tagging each triple with an identifier. However, this is hidden from the user who views the knowledge base as a set of "SPOTL" quintuples, where T is for time and L for location. The SPOTLX query language is used to access YAGO2. Although SPOTLX is a formal language, it is significantly easier to use than is SPARQL for queries involving time and location (which in SPARQL would require many joins for reified triplestores). SPOTLX does not accommodate quantification or negation, but can handle queries with prepositional aspects involving time and location. However, no mention is made of chained complex PPs.

Alexandria [13] is an event-based triplestore, with 160 million triples (representing 13 million n-ary relationships), derived from FreeBase. Alexandria uses a neo-Davidsonian [28] event-based semantics. In Alexandria, queries are parsed to a syntactic dependency graph, mapped to a semantic description, and translated to SPARQL queries containing named graphs. Queries with simple PPs are accommodated. However, no mention is made of negation, nested quantification, or chained complex PPs.

The systems referred to above have made substantial progress in handling ambiguity and matching NL query words to URIs. However, they appear to have hit a roadblock with respect to natural-language coverage. Most can handle simple PPs such as in "`who was born in 1918`" but none can handle chained complex PPs, containing quantifiers, such as "`in us_naval_observatory in 1877 or 1860`". There appear to be three reasons for this: 1) those NLQIs that were designed for non-reified triplestores, such as DBpedia, do not appear to be easily extended to reified triplestores that are necessary for complex PPs. 2) those NLQIs that were designed for non-reified or reified triplestores, and which translate the NL queries to SPARQL, suffer from the fact that SPARQL was originally designed for non-reified triplestores. Although SPARQL was extended to handle "named graphs" [25] which support a limited form of reification but appear to be suitable only for provenance data. SPARQL was also extended to accommodate triple identifiers. 3) The YAGO2 system is the only system that has an NLQI for a reified triplestore that does not translate to SPARQL. However, YAGO2 can only accommodate PPs related to time and location and does not support quantification.

Blackburn and Bos [24] implemented lambda calculus with respect to natural language, in Prolog, and [16] have extensively discussed such implementation in Haskell. Implementation of the lambda calculus for open-domain question answering has been investigated by [23]. The SQUALL query language [6, 10] is a Controlled Natural Language (CNL) for querying and updating triplestores represented as Resource Description Framework (RDF) graphs. SQUALL can return answers directly from remote triplestores, as we do, using simple SPARQL-endpoint triple

retrieval commands. It can also be translated to SPARQL queries which can be processed by SPARQL endpoints for faster computation of answers. SQUALL syntax and semantics are defined as a Montague Grammar facilitating the translation to SPARQL.SQUALL can handle quantification, aggregation, some forms of negation, and chained complex prepositional phrases. It is also written in a functional language. However, some queries in SQUALL require the use of variables and low-level relational algebraic operators (see for example, the queries on page 118 of [6]).

3.8 Concluding Comments

We have presented a compositional event semantics for computing the answers to English questions, and have shown how it can be used to query a remote event-based triplestore. We are currently working on three enhancements: 1) scaling up the NLQI to work with triplestores containing millions of events, and 2) increasing the coverage of English to accommodate negation, fusion events where roles can be assigned more than one value, and 3) modifying our approach of PPs to more consistently handle chained PPs.

Extensible NLQIs are necessary if the potential of the semantic web is realized and new data is added to existing triplestores by people who may not have been involved in the creation of those triplestores. We hope that other researchers who are familiar with Haskell will download and experiment with the software that we have developed; all of which is available through the links that we have provided. Our event-based triplestore is also available for remote access at the URL links that we have given.

Our semantics could be easily extended to accommodate very simple negation as in the query "`no moon orbits two planets`". However, the query "`which person orbits no moons?`" would not return the correct answer. The reason is that list returned by evaluating the denotation corresponding to "`orbits no moons`" would only contain entities that orbits something but not a moon. It would not contain entities that orbit nothing. This problem is related to how the "closed world as-

sumption" is implemented. To solve tis problem, we will begin by investigating the methods used in the experimental NLQIs Orakel [20], PANTO [22], Pythia [14], and TBSL [8]. We shall also consider theoretical computational linguistic approaches for dealing with negation and quantification in event-based semantics, e.g. [15].

In this paper, we have not addressed the problems that result from the user's lack of knowledge of the URIs used in the triplestores. Significant progress has been made by others, e.g. [12], in tackling this problem.

Also, we have not considered how our semantics can be automatically tailored for a particular triplestore. We shall begin by considering how Aqualog [21], PowerAqua [11] and Orakel [20] achieve portability with respect to the different ontologies used in different triplestores.

It should be noted that our current treatment of PPs is linguistically naive and suffers from problems with entailment (deriving logical consequences), as discussed by Partee [26], when certain kinds of prepositional or adverbial phrases are chained together. However, our proposed approach will accommodate many types of queries correctly, and the problem with entailment, which is also problematic for all existing triplestore NLQIs, will also be investigated in future work.

Acknowledgments

This research was funded by a grant from NSERC of Canada.

Bibliography

[1] R. Hafiz, R. Frost, S. Peelar, P. Callaghan, and E. Matthews. *The XSAIGA Package*. https://hackage.haskell.org/package/XSaiga, 2020 (cit. on p. 31).

[2] S. M. Peelar and R. A. Frost. "Accommodating Negation in an Efficient Event-based Natural Language Query Interface to the Semantic Web". In: *Proceedings of the 16th International Conference on Web Information Systems and Technologies, WEBIST 2020, Budapest, Hungary, November 3-5, 2020*. Ed. by

M. Marchiori, F. J. D. Mayo, and J. Filipe. SCITEPRESS, 2020, pp. 83–92. URL: https://doi.org/10.5220/0010148500830092 (cit. on p. 36).

[3] R. A. Frost and S. M. Peelar. "An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore". In: *Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, October 8th - 9th, 2018*. Ed. by K.-S. Choi, L. E. Anke, T. Declerck, D. Gromann, J.-D. Kim, A.-C. N. Ngomo, M. Saleem, and R. Usbeck. Vol. 2241. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 1–16. URL: http://ceur-ws.org/Vol-2241/paper-01.pdf (cit. on p. 18).

[4] S. Peelar. "Accommodating prepositional phrases in a highly modular natural language query interface to semantic web triplestores using a novel event-based denotational semantics for English and a set of functional parser combinators". MA thesis. University of Windsor (Canada), 2016 (cit. on pp. 24, 26).

[5] L. Champollion. "The interaction of compositional semantics and event semantics". In: *Linguistics and Philosophy* 38.1 (2015), pp. 31–66 (cit. on pp. 23, 27).

[6] S. Ferré. "SQUALL: a controlled natural language as expressive as SPARQL 1.1". In: *International Conference on Application of Natural Language to Information Systems*. Springer. 2013, pp. 114–125 (cit. on pp. 37, 38).

[7] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. "YAGO2: A spatially and Temporally Enhanced Knowledge Base from Wikipedia". In: *Artificial intell.* 194 (2013), pp. 28–61 (cit. on p. 36).

[8] K. Höffner, C. Unger, L. Bühmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, and P. Cimiano. "User Interface for a Template Based Question Answering

System". In: *Knowledge Engineering and the Semantic Web*. Springer, 2013, pp. 258–264 (cit. on p. 39).

[9] D. Davidson and G. Harman. *Semantics of natural language*. Vol. 40. Springer Science & Business Media, 2012 (cit. on p. 23).

[10] S. Ferre. "SQUALL: A Controlled Natural Language for Querying and Updating RDF Graphs". In: *proc. of CNL 2012*. LNCS 7427. 2012, pp. 11–25 (cit. on p. 37).

[11] V. Lopez, M. Fernández, E. Motta, and N. Stieler. "PowerAqua: Supporting Users in Querying and Exploring the Semantic Web". In: *Semantic Web* 3.3 (2012), pp. 249–265 (cit. on p. 39).

[12] S. Walter, C. Unger, P. Cimiano, and D. Bär. "Evaluation of a Layered Approach to Question Answering Over Linked Data". In: *The Semantic Web–ISWC 2012*. Springer, 2012, pp. 362–374 (cit. on p. 39).

[13] M. Wendt, M. Gerlach, and H. Düwiger. "Linguistic Modeling of Linked Open Data for Question Answering". In: *proc. of interact. with Linked Data (ILD 2012)[37]* (2012), pp. 75–86 (cit. on p. 37).

[14] C. Unger and P. Cimiano. "Pythia: Compositional Meaning Construction for Ontology-based Question Answering on the Semantic Web." In: *NLDB 2011, LNCS 6716*. 2011, pp. 153–160 (cit. on p. 39).

[15] L. Champollion. "Quantification in Event Semantics". In: *Talk given at the 6th int. symp. of cogn., Logic and commun.* 2010 (cit. on p. 39).

[16] J. Van Eijck and C. Unger. *Computational semantics with functional programming*. Cambridge University Press, 2010 (cit. on p. 37).

[17] J. Wheeler. "The hsparql Package". In: *The Haskell Hackage Repository*. http://hackage.haskell.org/package/hsparql-0.1.2. 2009 (cit. on p. 32).

[18] R. A. Frost, R. Hafiz, and P. Callaghan. "Parser Combinators for Ambiguous Left-Recursive Grammars". In: *Practical Aspects of Declarative Languages, 10th International Symposium, PADL 2008, San Francisco, CA, USA, January 7-8, 2008*. Ed. by P. Hudak and D. S. Warren. Vol. 4902. Lecture Notes in Computer Science. Springer, 2008, pp. 167–181. URL: https://doi.org/10.1007/978-3-540-77442-6%5C_12 (cit. on p. 33).

[19] S. Rothstein. *Structuring events: A study in the semantics of aspect*. Vol. 5. John Wiley & Sons, 2008 (cit. on p. 23).

[20] P. Cimiano, P. Haase, and J. Heizmann. "Porting Natural Language Interfaces Between Domains: an Experimental User Study With the ORAKEL System". In: *Proc. 12th Intnl. Conf. on intell. User Interfaces*. ACM. 2007, pp. 180–189 (cit. on pp. 36, 39).

[21] V. Lopez, V. Uren, E. Motta, and M. Pasin. "AquaLog: An Ontology-driven Question Answering System for Organizational Semantic Intranets". In: *Web semant.: sci., serv. and Agents on the World Wide Web* 5.2 (2007), pp. 72–105 (cit. on p. 39).

[22] C. Wang, M. Xiong, Q. Zhou, and Y. Yu. "Panto: A portable Natural Language Interface to Ontologies". In: *The Semantic Web: Research and appl.* Springer, 2007, pp. 473–487 (cit. on p. 39).

[23] K. Ahn, J. Bos, D. Kor, M. Nissim, B. L. Webber, and J. R. Curran. "Question Answering with QED at TREC 2005." In: *TREC*. 2005 (cit. on p. 37).

[24] P. Blackburn and J. Bos. "Representation and inference for natural language". In: *A first course in computational semantics. CSLI* (2005) (cit. on p. 37).

[25] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. "Named Graphs, Provenance and Trust". In: *proc. of the 14th int. conf. on World Wide Web*. ACM. 2005, pp. 613–622 (cit. on p. 37).

[26] B. H. Partee. "Formal Semantics". In: *Lectures at a workshop in Moscow. http://people. umass. edu/partee/RGGU_ 2005/RGGU05_formal_ semantics. htm*. 2005 (cit. on p. 39).

[27] F. Landman. "Plurality. Handbook of Contemporary Semantics". PhD thesis. ed. S. Lappin, 425- 457. Blackwell: Oxford, 1996 (cit. on p. 29).

[28] T. Parsons. *Events in the Semantics of English*. Vol. 5. Cambridge, Ma: MIT Press, 1990 (cit. on p. 37).

[29] D. Dowty, R. Wall, and S. Peters. *Introduction to Montague Semantics*. Dordrecht, Boston, Lancaster, Tokyo: D. Reidel Publishing Company, 1981 (cit. on pp. 19, 22, 23, 27).

[30] D. E. Knuth. "Semantics of context-free languages". In: *Mathematical systems theory* 2.2 (1968), pp. 127–145 (cit. on p. 32).

Chapter 4

A New Data Structure for Processing Natural Language Database Queries

This paper was published as:

R. A. Frost and S. M. Peelar. "A New Data Structure for Processing Natural Language Database Queries". In: *Proceedings of the 15th International Conference on Web Information Systems and Technologies, WEBIST 2019, Vienna, Austria, September 18-20, 2019*. 2019, pp. 80–87. URL: https://doi.org/10.5220/0008124300800087

It was featured at WEBIST 2019. This paper was nominated for the Best Student Paper Award and we were invited to submit an extended revised paper with new contributions for inclusion as a chapter in the WEBIST 2019 Springer Book.

4.1 Introduction

We begin by describing a Natural Language Query Interface (NLQI) that we have built. We hope that the interface will motivate readers to look into our modifications to Montague Semantics (MS) [17]. In Section 4.2, we explain how our NL

Query interface (NLQI) can be accessed through the Web. In Sub-section Section 4.2.1, we describe the Semantic Web triplestore. In Section 4.3 we discuss example queries and their results: in Section 4.3.2, we provide examples of what are often referred to as "non-compositional" features of NL that our NLQI can handle, and in Section 4.3.3 we give examples of NL structures that could be accommodated by extensions to our approach. With each of the examples we provide an informal explanation of how the answer is, or could be, computed.

In Section 4.4, we describe the new FDBR data structure which is central to our approach, and which can be created from an event-based triplestore (as we do in our online NLQI), or from a relational database.

Much of our semantics is based on MS. We differ in these ways:

1. We add events to the basic ontological concepts of entities and truth values.

2. Each event has a number of roles associated with it. Each role has an entity as a value.

3. For efficiency, we use sets of entities rather than characteristic functions of those sets as is the case in MS.

4. We define transitive n-ary verbs in terms of sets of events, each with n roles.

5. We compute FDBRs, the novel datastructure presented in this paper, from sets of events (could be computed from relations), and use them in the denotations of transitive verbs, and in computing results of queries containing prepositional phrases. Although not referred to as an FDBR, the use of relational images in denotations of verbs was first proposed by [16].

We hope that this paper reawakens an interest in Compositional Semantics, in particular for NL query processing.

4.2 How to Access our NLQI

Our NL interface can be accessed at the following web site:

`http://speechweb2.cs.uwindsor.ca/solarman4/demo_sparql.html`

4.2.1 The Triplestore that is Queried

Our NLQI computes answers with respect to a triplestore containing data about the planets, the moons that orbit them, and the people who discovered those moons, and when, where and with what implement they were discovered. Note that each set of triples associated with an event could be equally well be represented by a row in a relational database.

The triplestore contains triples such as the following which represent the event #1045 in which *hall* (in the role of "*subject*") discovered *phobos* (in the role of "*object*") in 1877 (in the role of "*year*") with the `refractor_telescope_1` (in the role of "*implement*") at the `us_naval_observatory` (in the role of "*location*").

Table 4.1: Events of type "Discover". The full URIs of the events, properties, and entities have been omitted here.

Event	Property	Entity
event1045	subject	hall
event1045	object	phobos
event1045	type	discover_ev
event1045	year	1877
event1045	location	us_naval_observatory
event1045	implement	refractor_telescope_1

Events representing set membership are represented as follows:

Table 4.2: Events of type "Membership".

Event	Property	Entity
event1128	subject	galileo
event1128	object	person
event1128	type	membership

The complete triplestore, which contains tens of thousands of triples, is hosted on a remote compute server using the Virtuoso software [7] and can be accessed by following the link at the beginning of Section 4.2.

4.3 Example Queries

Our NLQI can answer millions of queries with respect to the triplestore discussed above. The NLQI can accommodate queries containing common and proper nouns, adjectives, conjunction and disjunction, intransitive and transitive verbs, nested quantification, superlatives, chained prepositional phrases containing quantifiers, comparatives and polysemantic words. In the following we provide an informal explanation of how the answer is computed.

4.3.1 Queries Demonstrating the Range of NL Features that our NLQI can Accommodate

`phobos spins` ⇒ *True*

`phobos is a moon` ⇒ *True*

The function denoted by "`phobos`" checks to see if *phobos* is a member of the *spin* set, and secondly if *phobos* is a member of the *moon* set.

`a moon spins` ⇒ *True*

`every moon spins` ⇒ *True*

`an atmospheric moon exists` ⇒ *True*

The function denoted by "`a`" checks to see if the intersection of the *moon* set and *spin* set is non-empty. The function denoted by "`every`" checks to see if the set of the *moon* set is a subset of the *spins* set. The denotations of "`a`" and "`every`" that we use are set-theoretic event-based versions of the denotations from MS which uses characteristic functions. The answer to the third query is obtained by checking if the intersection of the *atmospheric* set and the *moon* set is non-empty.

`hall discovered` ⇒ *True*

All of the events of type "*discover*" are collected together and are checked to see if e_{hall} is found as the subject role value of any of them. If so, *True* is returned.

`when did hall discover` ⇒ *1877*

The "*year*" property of the events returned by "`hall discover`" (treated as "`hall discovered`") are returned.

`phobos was discovered` ⇒ *True*

All of the events of type "*discover*" are collected together and are checked to see if e_{phobos} is found as the object role value of any of them. If so *True* is returned.

`earth was discovered` ⇒ *False*

Earth was not discovered by anyone, according to our data.

`did hall discover phobos` ⇒ *True*

All of the events of type "*discover*" are collected together and are checked to create a pair (s, evs) for each value of the subject attribute found in the set of events. *evs* is the set of events to which the subject attribute is related through a discovery event. Each pair is then examined to see if the function denoted by the object termphrase (in this case *phobos*) returns a non-empty set when applied to a set (called an *FDBR*, which is described later) generated from the set of *evs* in the pair, and if so the subject of the pair is added to the set which is returned as the denotation of the verbphrase part of the query. The denotation of the termphrase at the beginning of the query is then applied to the denotation of the verbphrase to obtain the answer to the query.

Owing to the fact that our semantics is compositional the *subject* and *object* termphrases of the query above can be replaced by any termphrases. For example:

`a person or a team discovered every moon that orbits mars` ⇒ *True*

`who discovered two moons that orbit mars` ⇒ *hall*

"`who`", "`what`", "`where`", "`when`" and "`how`" can be used in place of the *subject* termphrase. Different role values are returned depending on which "wh.." word is used in the query.

`where discovered by galileo` ⇒ *padua*

`when discovered by galileo` ⇒ *1610*

`every telescope was used to discover a moon` ⇒ *True* (w.r.t.our data)

`a moon was discovered by every telescope` ⇒ *False*

`a telescope was used by hall to discover two moons` ⇒ *True*

`which moons were discovered with two telescopes`

⇒ *halimede laomedeia sao themisto*

`who discovered deimos with a telescope that was used to discover every moon that orbits mars` ⇒ *hall*

`who discovered a moon with two telescopes`

⇒ *nicholson science_ team_ 18 science_ team_ 2*

`how was sao discovered` ⇒ *blanco_ telescope canada-france-hawaii_ telescope*

`how discovered in 1877` ⇒ *refractor_ telescope_ 1*

`how many telescopes were used to discover sao` ⇒ *2*

`who discovered sao` ⇒ *science_ team_ 18*

`how did science_team_18 discover sao`

⇒ *blanco_ telescope canada-france-hawaii_ telescope*

`which planet is orbited by every moon that was discovered by two people` ⇒ *saturn; none*

`which person discovered a moon in 1877 with every telescope that was used to discover phobos` ⇒ *hall; none*

`who discovered in 1948 and 1949 with a telescope` ⇒ *kuiper*

4.3.2 Queries with "Non-Compositional" Structures

We agree with many other researchers that natural language has non-compositional features but believe that the non-compositionality is mostly problematic when the objective is to give a meaning to an NL expression without a context. It is less problematic when answering NL queries. As illustrated below, the person posing the query, or the database or triplestore can provide contexts that help resolve much of the ambiguity resulting from non-compositional features.

The advantages of a using a compositional semantics include 1) the answer to a query is as correct as the data from which it is derived, 2) the meaning of sub

phrases within a query can be discussed formally, 3) the query language can be extended such that all existing phrases maintain their original meanings, 4) the definition of syntax and semantics in the compositional semantics can be used as a blueprint for the implementation of the query processor.

Some researchers have provided examples of what they claim to be non-compositional structures in NL. For example, Hirst [14] gives the example of the verb "depart" which he states is not compositional because its meaning changes with the prepositional phrase(s) which follow it, and that the definition of compositionality needs to be modified to include the requirement that the function used to compose the meaning of parts must be systematic. We claim that our semantics for verbs is systematic as the denotations of subject and object termphrases, and the possibly empty list of prepositional phrases following the verb are treated equally and are all used in the same way to filter the set of events of the type associated with the verb, before that set is returned as the denotation of the verb phrase. This is illustrated in the following queries:

`who discovered` ⇒ *bernard bond cassini cassini_ imaging_ science_ team christy dollfus galileo etc...*

No *subject*, *object* or prepositional phrase is given in the query, and so all events of type "*discover*" are returned by the verbphrase and the denotation of the word "`who`" picks out the *subject*s from those events.

`where discovered io` ⇒ *padua*

No *subject*, or prepositional phrase is given in the query, and so all events of type "*discover*" are considered and filtered by the denotation of the *object* termphrase "`io`" and then, those that pass the filter are returned by the verbphrase and the word "`where`" picks out the location from those events.

`who discovered in 1610` ⇒ *galileo*

No *subject* or *object* is in the query so all events of type "*discover*" are considered and only those with attribute "*year*" equal to 1610 pass the filter and then the denotation

of the word "`who`" selects the subject which is returned.

In our semantics, the ***subject*** and ***object*** termphrases are treated as filters as are all prepositional phrases, as shown in the following example:

`who discovered every moon that orbits mars with one telescope or a moon that orbits Jupiter with a telescope` ⇒ *one. ; none. ; none. ; bernard galileo kowal melotte nicholson perrine science_team_1 science_team_2 ; hall ; hall ; none.*

Several results are returned because the query is syntactically ambiguous.

`where discovered in 1610` ⇒ *padua*

`how discovered in padua` ⇒ *galilean_telescope_1*

These queries retrieve the ***location*** and ***implement*** properties of the events of "`discovered in 1610`" and "`discovered in padua`" respectively.

4.3.3 Extensions to the Semantics

Some phrases containing nested quantifiers are given by some researchers, as examples of non-compositionality. For example: "a US diplomat was sent to every capital" is often read as having two meanings which can only be disambiguated by additional knowledge. We argue that the person posing a query can express the query unambiguously if they are familiar with quantifier scoping conventions used by our processor, as illustrated in the following:

`christy or science_team_19 or science_team_20 or science_team_21 discovered every moon that orbits pluto` ⇒ *False*

In our semantics, quantifier scoping is always leftmost/outermost, and an unambiguous query can be formulated as follows:

`every moon that orbits pluto was discovered by christy or science_team_19 or science_team_20 or science_team_21` ⇒ *True*

Some examples of non-compositionality involve polysemantic superlative words such

as "most" in, for example:

"Who discovered most moons that orbit P. Where P is a planet."

If "`most`" is treated as "more than half" then:

`who discovered most moons that orbit mars` ⇒ *hall*

Because our semantics currently allows only this reading. However, the answer to the alternate reading "who discovered the most moons that orbit *P*" – i.e. more than anyone else who discovered a moon that orbits *P*. Could be obtained in our semantics by comparing all of the (ent, *evs*) pairs returned by the verbphrase to see which ***subject*** is paired with most *objects*. We are currently working on this and other extensions to our semantics.

`how was every moon that orbits saturn discovered` ⇒ *cassini reflector_telescope_1 aerial_telescope_1 refractor_telescope_4 etc...*

It may be surprising that *cassini* is returned in the answer since it is not a *telescope*, but is instead a *spacecraft*. However, since it was used to discover at least one *moon* that orbits *saturn*, it is considered to have fulfilled the ***implement*** role and is encoded as such in the triplestore.

4.4 The FDBR: A Novel Data Structure for Natural Language Queries

4.4.1 Montague Semantics

All quantifiers, such as "`a`", "`every`" and "`more than two`" are treated in MS as functions which take two characteristic functions of sets as arguments and return a Boolean value as result. Our modifications to MS are to use sets of entities instead of predicates/characteristic functions of those sets, and to pair sets of events with each entity; the set of events paired with an entity justify the entity's inclusion in

the denotation. For example:

$$\|propernoun\| =$$
$$\lambda p.\{(e,evs) \mid (e,evs) \in p \ \&\ e = \text{the entity associated with the proper noun}\}$$

$$\|spins\| = \{(\mathrm{e}_{\mathrm{phobos}}, \{\mathrm{ev}_{1360}\}), (\mathrm{e}_{\mathrm{deimos}}, \{\mathrm{ev}_{1332}\}),\ etc \ldots\}$$

Therefore,

$$\|phobos\ spins\| = \|phobos\|\ \|spins\|$$
$$= \lambda s.\{(e,evs) | (e,evs) \in s\ \&\ e = \mathrm{e}_{\mathrm{phobos}}\}\{(\mathrm{e}_{\mathrm{phobos}}, \{\mathrm{ev}_{1360}\}), (\mathrm{e}_{\mathrm{deimos}}, \{\mathrm{ev}_{1332}\}), \ldots\}$$
$$= \{(\mathrm{e}_{\mathrm{phobos}}, \{\mathrm{ev}_{1360}\})\}$$

$$\|a\| = \lambda ms.\{(e_1, evs_2) \mid (e_1, evs_1)) \in m\ \&\ (e_2, evs_2) \in s\ \&\ e_1 = e_2\}$$

$$\|a\ moon\ spins\| = \{(\mathrm{e}_{\mathrm{phobos}}, \{\mathrm{ev}_{1360}\}),\ (\mathrm{e}_{\mathrm{deimos}}, \{\mathrm{ev}_{1332}\}),\ etc \ldots\}$$

Note that the events *evs* paired with the entities returned in the denotation of "`was every moon that orbits saturn discovered`" are the events representing membership of those entities of type "*moon*" in the object value of events of type "*discover*". This enables additional data to be accessed from those events, as illustrated in the last example query in the previous section.

4.4.2 The FDBR

In order to generate the answer to "`hall discovered every moon that orbits mars`", $\|every\|$ is applied to $\|moon\ that\ orbits\ mars\|$ (i.e. the set of *moons* that orbit *mars*), as first argument, and the set of entities that were discovered by *hall*, as the second argument. Our semantics generates this set from the set of events of type "*discover*" where the subject role is the entity associated with *hall*, as discussed below:

Every set of n-ary events (i.e. events with n roles) of a given type, e.g. discovery, defines $n^2 - n$ binary relations. For example, for discovery events:

$\text{discover}_{rel:subject\rightarrow object}$ $\text{discover}_{rel:subject\rightarrow year}$ $\text{discover}_{rel:subject\rightarrow implement}$ $\cdots$
$\text{discover}_{rel:object\rightarrow subject}$ $\text{discover}_{rel:object\rightarrow year}$ $\text{discover}_{rel:object\rightarrow implement}$ $\cdots$
$\text{discover}_{rel:year\rightarrow subject}$ $\text{discover}_{rel:year\rightarrow object}$ $\text{discover}_{rel:year\rightarrow implement}$ $\cdots$

etc... to 20 binary relations for the set of discovery events or an 5-ary discovery relation. For example:

$$\text{discover}_{rel:subject\rightarrow object} = \{(\text{ev}_{1045}, \text{e}_{\text{hall}}, \text{e}_{\text{phobos}}), (\text{ev}_{1046}, \text{e}_{\text{hall}}, \text{e}_{\text{deimos}}), etc\ldots\}$$

If we collect all of the values from the range of a relation that are mapped to by each value v from the domain (i.e. the image of v under the relation r) and create the set of all pairs $(v, image\_of\_v)$, we obtain a function defined by the relation r, i.e. the FDBR. For example:

$$\text{FDBR}(\text{discover}_{rel:subject\rightarrow object}) = \{(\text{e}_{\text{hall}}, \{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}),\ (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\}), etc\ldots\}$$

It is these functions that are created, and used, by the denotation of the transitive verb associated with the type of the events. For example in calculating the value of $\|who\ discovered\ every\ moon\ that\ orbits\ mars\|$, $\|every\|$ is applied to the set of entities which is the denotation of "`moon that orbits mars`" (i.e $\{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}), (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\}$) and all of the images that are in the second field of the pairs in $\text{FDBR}(\text{discover}_{rel:subject\rightarrow object})$.

For the pair $(\text{e}_{\text{hall}}, \{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}), (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\})$, $\|every\|$ returns the non-empty set $\{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}), (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\}$, and the value in the first field, i.e. e_{hall}, is subsequently returned with the answer to the query.

The various FDBRs are used to answer different types of queries. For example:

`who discovered phobos and deimos` $\Rightarrow$ *hall*

uses FDBR(discover$_{rel:subject \rightarrow object}$)

`where discovered by galileo` $\Rightarrow$ *padua*

uses FDBR(discover$_{rel:location \rightarrow subject}$)

`how discovered in 1610 or 1855` $\Rightarrow$ *galilean_ telescope_ 1*

uses FDBR(discover$_{rel:implement \rightarrow year}$)

4.5 Handling Prepositional Phrases

Prepositional phrases (PPs) such as "`with a telescope`" are treated similarly to the method above, except that the termphrase following the preposition is applied to the set of entities that are extracted from the set of events in the FDBR function, according to the role associated with the preposition. The result is a "filtered" FDBR which is further filtered by subsequent PPs.

4.6 Quantifiers and Events

In 2015, Champollion [2] stated that, at that time, it was generally thought by linguists that integration of Montagovian-style compositional semantics and Davidsonian–style event semantics [15, 18] was problematic, particularly with respect to quantifiers. Champollion did not agree with that analysis and presented an integration which he called "quantificational event semantics" which he claimed solved the difficulties of integration by assuming that verbs and their projections denote existential quantifiers over events and that these quantifiers always take lowest possible scope.

In this paper, we borrow much from Montague Semantics (MS), Davidsonian Event Semantics, and Champollion's Quantificational Event Semantics. However, we provide definitions of our denotations in the notation of set theory, which improves computational efficiency and, we believe, simplifies understanding of our denotations. We also believe that our semantics is intuitive, systematic, and compositional.

4.7 Our Approach with Relational Databases

Our NLQI could be easily adapted for use with conventional relational databases. Each row in a relation *Rel* can be thought of as representing an event of type *Rel*, and each column name can be thought of as a role name. The event itself would serve as the primary key, and only the triple retrieval function would need to be modified. This architecture allows the denotations to remain unchanged and yet still work with different types of databases.

4.8 Implementation of our NLQI

We built our query processor as an executable attribute grammar using the X-SAIGA Haskell parser-combinator library package [10]. The *collect* function which converts a binary relation to an FDBR is one of the most compute intensive parts of our implementation of the semantics. However, in Haskell, once a value is computed, it can be made available for future use. We have developed an algorithm to compute $\mathsf{FDBR}(rel)$ in $O(n \log n)$ time, where n is the number of pairs in *rel*. Alternatively, the FDBR functions can be computed and stored in a cache when the NLQI is offline. Our implementation is amenable to running on low power devices, enabling it for use with the Internet of Things. A version of our query processor exists that can run on a common consumer network router as a proof of concept for this application. The use of Haskell for the implementation of our NLQI has many advantages, including:

1. Haskell's "lazy" evaluation strategy only computes values when they are required, enabling parser combinator libraries to be built that can handle highly ambiguous left-recursive grammars in polynomial time.

2. The higher-order functional capability of Haskell allows the direct definition of higher-order functions that are the denotations of some English words and phrases.

3. The ability to partially apply functions of n arguments to 1 to n arguments allows the definition and manipulation of denotation of phrases such as "`every moon`", and "`discover phobos`".

4. The availability of the *hsparql* [9] Haskell package enables a simple interface between our semantic processor and SPARQL endpoints to our triplestores.

4.9 Related Work

Orakel [11] is a portable NLQI which uses a Montague-like grammar and a lambda calculus semantics. Our approach is similar in this respect. Queries are translated to an expression of first order logic enriched with predicates for query and numerical operators. These expressions are translated to SPARQL or F-Logic. Orakel supports negation, limited quantification, and simple prepositional phrases.

YAGO2 [4] is a semantic knowledge base containing reified triples extracted from Wikipedia, WordNet and GeoNames, representing nearly 0.5 billion facts. Reification is achieved by tagging each triple with an identifier. However, this is hidden from the user who views the knowledge base as a set of "SPOTL" quintuples, where T is for time and L for location. The SPOTLX query language is used to access YAGO2. SPOTLX can handle queries with prepositional aspects involving time and location. However, no mention is made of chained complex PPs.

Alexandria [6] is an event-based triplestore, with 160 million triples (representing 13 million n-ary relationships), derived from FreeBase. Alexandria uses a neo-Davidsonian [15] event-based semantics. In Alexandria, queries are parsed to a syntactic dependency graph, mapped to a semantic description, and translated to SPARQL queries containing named graphs. Queries with simple PPs are accommodated. However, no mention is made of negation, nested quantification, or chained complex PPs.

The systems referred to above have made substantial progress in handling ambiguity and matching NL query words to URIs. However, they appear to have hit a roadblock with respect to natural-language coverage. Most can handle simple PPs

such as in "`who was born in 1918`" but none can handle chained complex PPs, containing quantifiers, such as "`in us_naval_observatory in 1877 or 1860`".

Blackburn and Bos [13] implemented lambda calculus with respect to natural language, in Prolog, and [8] have extensively discussed such implementation in Haskell. Implementation of the lambda calculus for open-domain question answering has been investigated by [12]. The SQUALL query language [3, 5] is a controlled natural language (CNL) for querying and updating triplestores represented as RDF graphs. SQUALL can return answers directly from remote triplestores, as we do, using simple SPARQL-endpoint triple retrieval commands. It can also be translated to SPARQL queries which can be processed by SPARQL endpoints for faster computation of answers. SQUALL can handle quantification, aggregation, some forms of negation, and chained complex prepositional phrases. It is also written in a functional language. However, some queries in SQUALL require the use of variables and low-level relational algebraic operators (see for example, the queries on page 118 of [3]).

4.10 Concluding Comments

We are confident that, after we accommodate negation, our compositional semantics is appropriate for most queries that are likely to be asked of data stores containing knowledge related to household artifacts or domain specific information. The FDBR datastructure presented in this paper can be used to handle many kinds of complex language features, including chained prepositional phrases and superlatives. The way quantification is handled within the semantics is consistent with other work in this area, as discussed in Section 4.6. The approach chosen is flexible enough that it can accommodate queries to both relational and non-relational types of databases, including Semantic Web triplestores. It is also suitable for use on low power devices, which may be useful for applications on the Internet of Things (IoT).

In the future, we plan to scale up the capability of our NLQI further to access massive data stores such as DBpedia. To achieve this goal, we plan to accelerate the

FDBR generation process using specialized acceleration hardware, such as FPGAs and GPUs.

Bibliography

[1] R. A. Frost and S. M. Peelar. "A New Data Structure for Processing Natural Language Database Queries". In: *Proceedings of the 15th International Conference on Web Information Systems and Technologies, WEBIST 2019, Vienna, Austria, September 18-20, 2019*. 2019, pp. 80–87. URL: https://doi.org/10.5220/0008124300800087 (cit. on p. 44).

[2] L. Champollion. "The interaction of compositional semantics and event semantics". In: *Linguistics and Philosophy* 38.1 (2015), pp. 31–66 (cit. on p. 55).

[3] S. Ferré. "SQUALL: a controlled natural language as expressive as SPARQL 1.1". In: *International Conference on Application of Natural Language to Information Systems*. Springer. 2013, pp. 114–125 (cit. on p. 58).

[4] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. "YAGO2: A spatially and Temporally Enhanced Knowledge Base from Wikipedia". In: *Artificial intell.* 194 (2013), pp. 28–61 (cit. on p. 57).

[5] S. Ferre. "SQUALL: A Controlled Natural Language for Querying and Updating RDF Graphs". In: *proc. of CNL 2012*. LNCS 7427. 2012, pp. 11–25 (cit. on p. 58).

[6] M. Wendt, M. Gerlach, and H. Düwiger. "Linguistic Modeling of Linked Open Data for Question Answering". In: *proc. of interact. with Linked Data (ILD 2012)[37]* (2012), pp. 75–86 (cit. on p. 57).

[7] O. Erling and I. Mikhailov. "Virtuoso: RDF support in a native RDBMS". In: *Semantic Web Information Management*. Springer, 2010, pp. 501–519 (cit. on p. 46).

[8] J. Van Eijck and C. Unger. *Computational semantics with functional programming*. Cambridge University Press, 2010 (cit. on p. 58).

[9] J. Wheeler. "The hsparql Package". In: *The Haskell Hackage Repository*. http://hackage.haskell.org/package/hsparql-0.1.2. 2009 (cit. on p. 57).

[10] R. A. Frost, R. Hafiz, and P. Callaghan. "Parser Combinators for Ambiguous Left-Recursive Grammars". In: *Practical Aspects of Declarative Languages, 10th International Symposium, PADL 2008, San Francisco, CA, USA, January 7-8, 2008*. Ed. by P. Hudak and D. S. Warren. Vol. 4902. Lecture Notes in Computer Science. Springer, 2008, pp. 167–181. URL: https://doi.org/10.1007/978-3-540-77442-6%5C_12 (cit. on p. 56).

[11] P. Cimiano, P. Haase, J. Heizmann, and M. Mantel. *Orakel: A portable natural language interface to knowledge bases*. Tech. rep. Technical report, Institute AIFB, University of Karlsruhe, 2007 (cit. on p. 57).

[12] K. Ahn, J. Bos, D. Kor, M. Nissim, B. L. Webber, and J. R. Curran. "Question Answering with QED at TREC 2005." In: *TREC*. 2005 (cit. on p. 58).

[13] P. Blackburn and J. Bos. "Representation and inference for natural language". In: *A first course in computational semantics. CSLI* (2005) (cit. on p. 58).

[14] G. Hirst. *Semantic interpretation and the resolution of ambiguity*. Cambridge University Press, 1992 (cit. on p. 50).

[15] T. Parsons. *Events in the Semantics of English*. Vol. 5. Cambridge, Ma: MIT Press, 1990 (cit. on pp. 55, 57).

[16] R. Frost and J. Launchbury. "Constructing natural language interpreters in a lazy functional language". In: *The Computer Journal* 32.2 (1989), pp. 108–121 (cit. on p. 45).

[17] D. Dowty, R. Wall, and S. Peters. *Introduction to Montague Semantics*. Dordrecht, Boston, Lancaster, Tokyo: D. Reidel Publishing Company, 1981 (cit. on p. 44).

[18] D. Davidson. "The logical form of action sentences". In: (1967) (cit. on p. 55).

Chapter 5

A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore

This paper was published as:

S. M. Peelar and R. A. Frost. "A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore". In: *IEEE 14th International Conference on Semantic Computing, ICSC 2020, San Diego, CA, USA, February 3-5, 2020*. IEEE, 2020, pp. 257–262. URL: https://doi.org/10.1109/ICSC.2020.00054

It was featured at IEEE ICSC 2020. We were invited to submit a full length paper to Advances in Science, Technology and Engineering Systems Journal (ASTESJ) for a Special Issue on Multidisciplinary Sciences and Engineering. We have not submitted anything to the journal at this time.

5.1 Introduction

Many Natural Language Query Interfaces to relational databases and semantic web triplestores convert the NL query to a formal query language such as SQL or SPARQL and then execute the formal query with respect to the relational database or semantic web triplestore respectively. One problem with these approaches is that the interface is restricted by the difficulty of translating complex NL phrases to the formal query language. In particular, queries with chained prepositional phrases containing quantifiers have been difficult to accommodate. Examples of such queries are: "`who discovered two moons with a telescope in 1877 at us_naval_observatory`", and "`where was a telescope used by hall to discover phobos`"

An alternative approach is to treat the NL query as an expression of the lambda calculus, using an extended form of the denotational semantics of Richard Montague [25], and to calculate the answer directly by interpreting the lambda calculus expression with respect to the data store. All that are required to be extracted from the data store are unary relations corresponding to sets of entities associated with the denotations of common nouns, adjectives and intransitive verbs, and the $n^2 - n$ binary relations associated with n-ary transitive verbs.

Montague semantics can be easily implemented in a pure functional programming language such as Haskell. The higher-order functional capability of Haskell and the ability to partially apply higher-order functions enable Montague denotations of words such as "`every`" to be directly defined in the language. Furthermore, the availability of parser combinator libraries enables the construction of Executable Attribute Grammars (EAGs) within the language. This lends itself to a direct implementation of Montague-style integration of syntax rules with semantic rules.

5.2 A Prototype System

The pure functional programming language Haskell was used to build a prototype NL interface to a semantic web triplestore. It can be accessed at this URL:

`http://speechweb2.cs.uwindsor.ca/solarman4/demo_sparql.html`

Notably, our interface is able to accommodate highly complex chained prepositional phrases in queries, including those with superlatives. For example, our system can accommodate the query: "`who discovered the most vacuumous moons using the most telescopes in the most places`." A comprehensive list of examples can be found at the "More Examples" link at the URL above.

If a syntactically ambiguous query is entered, the results from each possible interpretation are returned, along with their corresponding syntax trees. For example, "`who discovered the most vacuumous moons in 1877`" could be treated as "`who (discovered (the most (vacuumous moons)) [in 1877])`". This style of notation was chosen to closely mirror how the semantics are internally evaluated in the Haskell language. Both parentheses and brackets denote scoping, and brackets denote lists of prepositional phrases

5.3 Related Work

Orakel [17] is a portable NLQI which uses a Montague-like grammar and a lambda-calculus semantics to analyze queries. The approach described in this paper is similar to Orakel in this respect. However, in Orakel, queries are translated to an expression of first-order logic enriched with predicates for query and numerical operators. These expressions are translated to SPARQL or F-Logic. Orakel supports negation, limited quantification, and simple prepositional phrases. Portability is achieved by having the lexicon customized by people with limited linguistic expertise. It is claimed that Orakel can accommodate n-ary relations with $n \geq 2$.

However, no examples are given of such queries being translated to SPARQL.

YAGO2 [9] is a semantic knowledge base containing reified triples extracted from Wikipedia, WordNet and GeoNames, representing nearly 0.5 billion facts. Reification is achieved by tagging each triple with an identifier. However, this is hidden from the user who views the knowledge base as a set of "SPOTL" quintuples, where T is for time and L for location. The SPOTLX query language is used to access YAGO2. Although SPOTLX is a formal language, it is significantly easier to use than is SPARQL for queries involving time and location (which in SPARQL would require many joins for reified triplestores). SPOTLX does not accommodate quantification or negation, but can handle queries with prepositional aspects involving time and location. However, no mention is made of chained complex PPs.

Alexandria [11] is an event-based triplestore, with 160 million triples (representing 13 million n-ary relationships), derived from FreeBase. Alexandria uses a neo-Davidsonian [23] event-based semantics. In Alexandria, queries are parsed to a syntactic dependency graph, mapped to a semantic description, and translated to SPARQL queries containing named graphs. Queries with simple PPs are accommodated. However, no mention is made of negation, nested quantification, or chained complex PPs.

The systems referred to above have made substantial progress in handling ambiguity and matching NL query words to URIs. However, they appear to have hit a roadblock with respect to natural-language coverage. Most can handle simple PPs such as in "`who was born in 1918`" but none can handle chained complex PPs, containing quantifiers, such as "`in us_naval_observatory in 1877 or 1860`". There appear to be three reasons for this: 1) those NLQIs that were designed for non-reified triplestores, such as DBpedia, do not appear to be easily extended to reified triplestores that are necessary for complex PPs. 2) those NLQIs that were designed for non-reified or reified triplestores, and which translate the NL queries to SPARQL, suffer from the fact that SPARQL was originally designed for non-reified triplestores. Although SPARQL was extended to handle "named graphs" [21] which support a limited form of reification but appear to be suitable only for provenance

data. SPARQL was also extended to accommodate triple identifiers. 3) The YAGO2 system is the only system that has an NLQI for a reified triplestore that does not translate to SPARQL. However, YAGO2 can only accommodate PPs related to time and location and does not support quantification.

Reference [20] implemented lambda calculus with repect to natural language, in Prolog, and [14] have extensively discussed such implementation in Haskell. Implementation of the lamda calculus for open-domain question answering has been investigated by [19]. The SQUALL query language [7, 10] is a controlled natural language (CNL) for querying and updating triplestores represented as RDF graphs. SQUALL can return answers directly from remote triplestores, as the approach described in this paper does, using simple SPARQL-endpoint triple retrieval commands. It can also be translated to SPARQL queries which can be processed by SPARQL endpoints for faster computation of answers. SQUALL syntax and semantics are defined as a Montague Grammar facilitating the translation to SPARQL.SQUALL can handle quantification, aggregation, some forms of negation, and chained complex prepositional phrases. It is also written in a functional language. However, some queries in SQUALL require the use of variables and low-level relational algebraic operators (see for example, the queries on page 118 of [7]).

5.4 The Extension to Montague Semantics

MS [25] defines the meaning of words, phrases, sentences and queries in terms of a space of functions that is built over a set of entities (the universe of discourse) and the Boolean values *True* and *False*. For example, the word "`moon`" denotes the characteristic function (logical predicate) which maps entities to *True* or *False*. The result is *True* if the entity is a moon, and *False* otherwise. One of Montague's many insights was his recognition that proper nouns such as "`phobos`" do not denote entities; rather they denote functions that take characteristic functions as argument and return Boolean values as result. For example "`phobos`" denotes the function $\lambda f.f\ \mathrm{e}_{\mathrm{phobos}}$ where $\mathrm{e}_{\mathrm{phobos}}$ represents the entity associated with the name Phobos.

For readers not familiar with the Lambda Calculus, the expression $\lambda f.expr$ is the name (and definition) of a function which, when applied to an argument g returns as result the expression *expr* with all instances of f in *expr* replaced by g. According to MS, the phrase "`phobos spins`" is interpreted as shown below, where $a \Longrightarrow b$ indicates that b is the result of evaluating a, $\|a\|$ represents the denotation (meaning) of a, w_{pred} is the logical predicate associated with the word w, and λ is the Lambda symbol.

$$
\begin{aligned}
& \|phobos\ spins\| \\
\Longrightarrow\ & \|phobos\|\ \|spins\| \\
\Longrightarrow\ & \lambda f.f\ \mathrm{e}_{\mathrm{phobos}}\ \|spins\| \\
\Longrightarrow\ & \lambda f.f\ \mathrm{e}_{\mathrm{phobos}}\ \mathrm{spins}_{pred} \\
\Longrightarrow\ & \mathrm{spins}_{pred}\ \mathrm{e}_{\mathrm{phobos}} \\
\Longrightarrow\ & True
\end{aligned}
$$

owing to the fact that Phobos does spin.

Montague's treatment of quantifiers such as "`a`", "`every`", "`some`", "`one`" etc. is to treat their denotations as higher-order functions. For example, the word "`every`" denotes the following function:

$$\|every\| = \lambda pq.(\forall x)\ (p\ x \Rightarrow q\ x)$$

For example:

$$
\begin{aligned}
& \|every\ moon\ spins\| \\
\Longrightarrow\ & (\|every\ moon\|)\ \|spins\| \quad \text{(from syntactic parsing)} \\
\Longrightarrow\ & (\lambda pq.(\forall x)\ (p\ x \Rightarrow q\ x)\ \mathrm{moon}_{pred})\ \mathrm{spins}_{pred} \\
\Longrightarrow\ & (\lambda q.(\forall x)\ (\mathrm{moon}_{pred}\ x \Rightarrow q\ x))\ \mathrm{spins}_{pred} \\
\Longrightarrow\ & (\forall x)\ \mathrm{moon}_{pred}\ x \Rightarrow \mathrm{spins}_{pred}\ x \\
\Longrightarrow\ & True \text{ (every moon in the universe of discourse spins)}
\end{aligned}
$$

5.4.1 A Computationally Tractable Version of Montague Semantics

The direct implementation of MS is not practical for applications with a large universe of discourse owing to the use of characteristic functions. For example, the denotation of "`every`" given above is computationally intractable in a query such as "`does every moon spin`" as it would require the characteristic function that is the denotation of "`moon`" to be applied to every entity in the universe of discourse. A more efficient alternative approach is to use the sets defined by characteristic functions directly in denotations [22, 24]. For example:

$$\|moon\| = \{\mathrm{e}_{\mathrm{phobos}}, \mathrm{e}_{\mathrm{deimos}}, \ldots\}$$

All other denotations are modified accordingly. For example:

$$\|phobos\| = \lambda s.\mathrm{e}_{\mathrm{phobos}} \in s \text{ (where } \in \text{ is the set membership operator)}$$
$$\|every\| = \lambda st.(s \subseteq t) \text{ (where } s \subseteq t \text{ returns } \textit{True} \text{ if } s \text{ is a subset of } t)$$
$$\|spins\| = \text{the set of entities that spin}$$

Thus the phrase "`every moon spins`" is interpreted as follows:

$$\|every\ moon\ spins\|$$
$$\Longrightarrow (\lambda st.(s \subseteq t))\ \|moon\|\ \|spins\|$$
$$\Longrightarrow \|moon\| \subseteq \|spins\|$$
$$\Longrightarrow \textit{True} \quad \text{(because all moons in the universe of discourse are in the set of things that spin)}$$

The phrase "`phobos spins`" is interpreted as follows:

$$\begin{aligned} & \|phobos\ spins\| \\ \Longrightarrow\ & (\lambda s.\mathrm{e}_{\mathrm{phobos}} \in s)\ \|spins\| \\ \Longrightarrow\ & \mathrm{e}_{\mathrm{phobos}} \in \|spins\| \\ \Longrightarrow\ & \mathit{True} \quad \text{(because Phobos, in the universe of} \\ & \qquad\qquad \text{discourse, is in the set of entities that spin)} \end{aligned}$$

5.4.2 An Event-Based Version of Montague Semantics

An event-based version of MS is needed to accommodate queries executed with respect to event-based reified triplestores [4]. Such a semantics has been developed and is described in earlier papers [4, 5] and in Peelar's Master's thesis [3].

It should be noted that binary relations can be obtained from event-based triplestores by first retrieving all triples of the type associated with the transitive verb, then extracting all event identifiers from those triples, followed by retrieving all subjects and objects associated with those events. The binary relation is obtained by pairing each subject with each object with which it is associated through an event.

In the event-based approach, rather than returning sets of entities as results of evaluating denotational expressions, sets of pairs are returned. Each pair $(\mathsf{ent}, \mathit{evs})$ consists of an entity ent paired with a set of events *evs* which justify the entity being in the answer. For example:

$$\|phobos\ spins\| \Longrightarrow \{(\mathrm{e}_{\mathrm{phobos}}, \{\mathrm{ev}_{1360}\})\}$$

where ev_{1360} is the event identifier for the event in which $\mathrm{e}_{\mathrm{phobos}}$ became a member of the set of entities which spin.

5.5 Denotations of Transitive Verbs

The main contribution of this paper and the associated demo program beyond the results described in [5, 13] is that Montague's treatment of transitive verbs is extended by using the $(n^2 - n)$ functions that are defined by the n-ary relation associated with each of the more complex transitive verbs. First Montague's treatment of transitive verbs will be introduced.

5.5.1 Montague's Treatment of Transitive Verbs

Transitive verbs in MS are handled using syntactic manipulation rather than with an explicit semantic denotation (see page 216 of [25]).

5.5.2 An Alternative Treatment of Transitive Verbs I

In earlier work [18], an explicit denotation for transitive verbs was developed that gives the same result as MS for some queries when their translations to lambda expressions are rewritten to their canonical forms. However, this approach does not work for queries such as "`hall discovered a moon`", since the denotation of the term-phrase "`a moon`" is more complex than the denotation of the term-phrase "`phobos`".

5.5.3 An Alternative Treatment of Transitive Verbs II

One solution to the problem above is to use sets rather than characteristic functions (predicates) of those sets (as discussed in Section 5.4.1) in the denotations of transitive verbs. The basic idea [24] which is adopted in this paper is to consider transitive verbs as relations from the subjects or objects of those verbs to the events they participate in. Specifically, transitive verbs are denoted using the *function defined by the binary relation* (FDBR) [3] induced by the relation *rel* that underlies

the verb:

$$\text{FDBR}(\mathit{rel}) = \{(x, \text{image}_x) \mid (\exists e)\ (x,e) \in \mathit{rel}\ \&\ \text{image}_x = \{y \mid (x,y) \in \mathit{rel}\}\}$$

Briefly, FDBR(rel) converts *rel* into a function without any loss of information by grouping together elements in the codomain that are related under the same element in the domain. For example, consider the relation underlying the active voice of "`discover`", $\text{discover}_{\mathit{rel}}$:

$$\text{FDBR}(\text{discover}_{\mathit{rel}}) = \{(\text{e}_{\text{hall}}, \{\text{ev}_{1045}, \text{ev}_{1046}\}),\ \text{etc}\ldots\}$$

If the transitive verb is followed by a term-phrase such as "`phobos`" or "`a moon`", then the function denoted by that term-phrase is used to "filter" the denotation of the transitive verb for relevant actors. For example:

$$\|\mathit{discovered}\|\ \|\mathit{phobos}\| \implies \{(\text{e}_{\text{hall}}, \{\text{ev}_{1045}\})\}$$

Similarly,

$$\|\mathit{discovered}\|\ \|\mathit{a\ moon}\| \implies \{(\text{e}_{\text{hall}}, \{\text{ev}_{1045}, \text{ev}_{1046}\}), \ldots\}$$

The denotation of the transitive verb "`discover`" follows from the above:

$$\begin{aligned}\|\mathit{discover}\| = \lambda t.\{(s,\ \mathit{relevs}) \mid (s, \mathit{evs})\ \in\ &\text{FDBR}(\text{discover}_{\mathit{rel}}) \\ &\&\ (t\ \text{obj\_fdbr}(\mathit{evs}) \neq \emptyset) \&\ \mathit{relevs} = \text{gather}(\text{obj\_fdbr}(\mathit{evs}))\}\end{aligned}$$

where obj_fdbr(evs) is the FDBR from the objects in the events of the set *evs* to the events they participate in within *evs*. In the examples above,

$$\text{obj\_fdbr}(\{\text{ev}_{1045}, \text{ev}_{1046}\}) = \{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}), (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\}$$

When $\|phobos\|$ is applied to $\text{obj\_fdbr}(\{\text{ev}_{1045}, \text{ev}_{1046}\})$, it filters the FDBR to only contain *relevant* pairs[1]. If the FDBR is empty after filtering, then the pair corresponding to that FDBR is discarded. The function $\text{gather}(fdbr)$ returns the set of all events in the second column of *fdbr*:

$$\text{gather}(fdbr) = \{ev \mid (\exists e)(\exists evs)\ (e, evs) \in fdbr\ \ \&\ \ ev \in evs\}$$

As another example, consider $\|discover\ every\ moon\|$:

$$\begin{aligned}
&\|discover\ every\ moon\| \\
\Longrightarrow\ &\|discover\|\ (\|every\ moon\|) \\
\Longrightarrow\ &(\lambda t.\{(s, relevs) \mid (s, evs)\ \in \text{FDBR}(\text{discover}_{rel}) \\
&\qquad \&\ t\ \ \text{obj_fdbr}(evs) \neq \emptyset \\
&\qquad \&\ \ relevs = \text{gather}(\text{obj_fdbr}(evs))\}) \\
&\quad (\|every\ moon\|) \\
\Longrightarrow\ &\{(s, relevs) \mid (s, evs)\ \in \text{FDBR}(\text{discover}_{rel}) \\
&\qquad \&\ \|every\ moon\|\ \ \text{obj_fdbr}(evs) \neq \emptyset \\
&\qquad \&\ \ relevs = \text{gather}(\text{obj_fdbr}(evs))\} \\
\Longrightarrow\ &\emptyset \qquad \text{(owing to the fact that no entity in the} \\
&\qquad\qquad \text{universe of discourse discovered every moon)}
\end{aligned}$$

Specifically, observe how the pair $(\text{e}_{\text{hall}}, \{\text{ev}_{1045}, \text{ev}_{1046}\}) \in \text{FDBR}(\text{discover}_{rel})$ is treated above:

$$\begin{aligned}
&\|every\ moon\|\ \ \text{obj_fdbr}(\{\text{ev}_{1045}, \text{ev}_{1046}\}) \\
\Longrightarrow\ &\|every\ moon\|\ \{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}), (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\} \\
\Longrightarrow\ &\emptyset \quad \text{(owing to the fact that there exist entities} \\
&\qquad \text{other than } \text{e}_{\text{phobos}} \text{ and } \text{e}_{\text{deimos}} \text{ in } \text{moon}_{FDBR})
\end{aligned}$$

[1]The notion of event relevance is discussed in more detail in [3]

In this event-based approach, the result of a verb-phrase such as "`discovered phobos`" is a list of pairs, each pair consisting of an entity which discovered phobos, paired with the set of events of type *discover* in which that entity was the subject and $\mathsf{e}_{\text{phobos}}$ was the object. In other words, the set of events in each pair can be thought of as a form of justification for the subject entity in the first field of that pair belonging in the result.

5.5.4 Accommodating Chained Prepositional Phrases

The above approach can be extended to support prepositional phrases in queries with only minor changes [3, 8]. Briefly, the denotations of the prepositions act as filters to the denotation of the transitive verb they apply to. Consider, for example, the query "`discovered in 1877 with a telescope`". In this query, the prepositions are "`in 1877`" and "`with a telescope`". Performing this filtering is identical to the term-phrase filtering shown above, with some added logic to select columns other than the subject and object from the relation.

The denotation for a preposition applied to a term-phrase is a pair (*props*, *tmph*) where *props* is a set of properties that an FDBR should be constructed from (for example "*implement*", "*year*", or "*location*"), and *tmph* is the term-phrase that will be applied to that resulting FDBR.

$$\begin{aligned}
\|at\| &= \lambda t.(\{location\},t) \\
\|in\| &= \lambda t.(\{location, year\},t) \\
\|with\| &= \lambda t.(\{implement\},t) \\
\|using\| &= \lambda t.(\{implement\},t) \\
\|to\| &= \lambda t.(\{subject\},t)
\end{aligned}$$

Under this approach, a prepositional phrase is treated in the same way as the term-phrase following the verb [3]. This approach is powerful enough that the word "`by`", as in "`discovered by`", can be treated in the same way as a preposition, enabling active and passive voices of transitive verbs to be treated in a uniform way. This

was one of the key contributions described in Peelar's Master's thesis [3].

$$\|by\| = \lambda t.(\{subject\}, t)$$

The denotation of the transitive verb is filtered in the order that the prepositions appear. To achieve this, the previous denotation for "`discover`" is modified such that $\mathsf{obj\_fdbr}(evs)$ is replaced with the more general $\mathsf{prop\_fdbr}(prop, evs)$, and the filtered FDBR from each applied termphrase is fed into the next preposition. If the passive form of the transitive verb is used, then the relation is flipped and the same logic applies.

5.5.5 Formal Denotations of Transitive Verbs

While a denotation for transitive verbs that allows for chained prepositional phrases and a unified treatment of active and passive voices improves expressibility, there are still a number of queries with transitive verbs that are not possible with the above approach. For example, consider the transitive verb "`used`", as in the query "`refractor_telescope_1 was used to discover phobos`". Here, the subject of the query is a refractor telescope, however it is neither the subject nor object of the relation underlying the denotation of "`used`", which here is the same relation underlying the denotation of "`discover`" – it is an implement in that relation. A relation would need to be constructed from the "*implement*" property of the relation rather than the "*subject*" or "*object*" property as in the denotations in Section 5.5.3 in order to compute this.

This can be addressed with minor modifications to those denotations. Namely, the underlying relation is generalized to be n-ary. The columns of the n-ary relation are the properties of the event type of the transitive verb, with one column corresponding to the event identifier itself as in the denotation in Section 5.5.3. A new function *make_binrel* is introduced which converts an n-ary relation r into a binary

relation by selecting two columns c_1 and c_2 from it:

$$\text{make\_binrel}(r, c_1, c_2) = \ldots$$

The FDBR is constructed from the underlying relation by using a new function that works on n-ary relations:

$$\text{FDBR}_{prop}(r) = \textbf{FDBR}(\text{make\_binrel}(r, prop, eventid))$$

Here, *eventid* refers to the column of the relation that contains the event. The denotation for transitive verbs is augmented with the property *prop* used to form the binary relation from the n-ary relation, for example "*subject*", "*object*", or "*implement*", replacing $\textbf{FDBR}(r)$ with $\text{FDBR}_{prop}(r)$. With these revisions, it is possible to express a denotation for the passive voice of the verb "`used`" (as in "`what was used by hall`"):

$$\begin{aligned} \|used\ by\| = \lambda t.\{(s,\ relevs) \mid (s, evs) \in \text{FDBR}_{implement}(\text{discover}_{rel}) \\ \& \ t\ \text{prop\_fdbr}(subject, evs) \neq \emptyset \\ \& \ relevs = \text{gather}(\text{obj\_fdbr}(evs))\} \end{aligned}$$

Now, denotations for transitive verbs can be provided from any property to any other property, for example from implements to years, or from years to objects. This could be useful for constructing NL interfaces for other languages, such as French.

5.6 The $n^2 - n$ Functions Defined By An n-ary Relation

The major contribution of this paper is to use the approach described above to considerably broaden the coverage of English compared to other systems and earlier [3, 4] query interfaces developed in this direct line of research. First, notice that the

Table 5.1: The "Discover" Relation

subject	object	date	implement	location
...	...	...	...	...
hall	phobos	1877	refractor_telescope_1	us_naval_observatory
...	...	...	...	...

phrase "`discover` *x*" often appears in contexts where the result expected is the set of subjects who discovered "*x*". However the words "`discover`" and "`discovered`" also appear in other contexts. For example, "`discover with a telescope`" (subjects expected), "`discovered by hall`" (objects expected), "`how was` *x* `discovered`" (implements expected), "`who used a telescope to discover something in 1877`" (subjects expected), and "`when was a telescope used to discover`" (years expected).

It has been observed that different functions can be defined for a set of events. Suppose that the event is thought of as a row in an *n*-ary relation, such as the 5-ary discover relation (Table 5.1).

The example in Section 5.5.1 used the FDBR from subjects to objects. However, there are $n^2 - n = (25 - 5)$ functions that can be defined from the 5-ary relation above if the function from a column to itself is excluded: subject to object, object to subject, subject to date, date to subject, date to subject, date to object etc. These functions can be used to answer any query about the discover relation, including those containing chained complex prepositional phrases.

5.7 Applicability to Relational Databases

An event-based triplestore can be thought of as a set of tables in a relational database where each event corresponds to a row in a table. Each table corresponds to a relation in the event-based triplestore (e.g, the "discover" relation). The event identifier, which is unique, becomes the row index. The only interface that the semantics has with the underlying database is through simple retrieval functions. Currently, they are implemented using Triple Pattern Fragments [6], but they could

also be implemented using simple SQL queries. Therefore, it is possible to use our approach with any relational database platform.

5.8 Implementation

NLQIs using Montague-type semantics are ideally suited for implementation in syntax-directed interpreters. One form of syntax-directed interpreter is an EAG in which the executable code of the interpreter has a close similarity to textbook attribute grammar notation. Accordingly, the Solarman NLQI is built as an executable specification of an attribute grammar using the Haskell X-SAIGA context-free parser combinator library [13, 15]. The source code for Solarman, including the X-SAIGA parser combinator library, is available on the Hackage package archive [1].

5.9 Future Work

5.9.1 Non-event based triplestores

Our approach requires an event-based triplestore to handle chained prepositional phrases. There is a clear need to support non-event-based Semantic Web triplestores as well. It may be possible to build an adapter from conventional triplestores to event-based triplestores using Semantic Web OWL schemas or a Machine Learning (ML) approach.

5.9.2 Very large triplestores

This approach currently is only viable for databases with tens of thousands of facts, whereas Semantic Web triplestores such as DBPedia [16] could contain millions. It is possible to memoize the semantics to avoid FDBRs from being re-computed throughout an expression, greatly improving the evaluation speed. The demonstration in Section 5.2 features an early version of this functionality enabling queries

with deeply nested transitive verbs to be efficiently evaluated (e.g, "`who discovered a moon that orbits a planet that is orbited by a thing`"). Our approach, which we will expand on in the future, lifts the semantics into a memoized form while maintaining their underlying compositionality. Furthermore, the FDBRs can be generated and cached offline instead of generating them on the fly.

5.9.3 Negation

Negation in general only holds if the *Closed World Assumption* can be satisfied. Informally, the Closed World Assumption can be characterized by the statement:

"*The absence of evidence can be assumed as being evidence of absence*".

The *Open World Assumption* on the other hand assumes the converse of the statement above. For example, if a particular entity p is not explicitly stated as being a member of the "person" set, then under the Closed World Assumption it can be assumed that p is not a member of that set. On the other hand, under the Open World Assumption, the only way to conclude p is not a member of the "person" set is if it is explicitly stated in the database.

The semantics described in this paper use the Open World Assumption and hence do not support negation. Work has been done on event-based semantics that support negation [12] in environments where the Closed World Assumption holds. It may be possible to use similar techniques to provide a drop-in denotation for "`not`" and "`no`", supporting negation in our semantics as well.

5.10 Conclusion

The approach described in this paper extends previous work on building natural-language query interfaces to online data stores by providing an explicit Montague-style efficient denotation for transitive verbs; and an approach for accommodating queries containing chained complex prepositional phrases. The viability of this approach was demonstrated by building an NL query interface to an event-based semantic-web triplestore containing thousands of facts. The approach could be used

with relational databases by considering the n^2n functions defined by each n-ary relation associated with n-ary transitive verbs. Research on scaling this approach is ongoing, with the goal being to create an interface to query data stores containing millions of facts.

Acknowledgment

This research was funded by a grant from the Natural Sciences and Engineering Research Council of Canada.

Bibliography

[1] R. Hafiz, R. Frost, S. Peelar, P. Callaghan, and E. Matthews. *The XSAIGA Package*. https://hackage.haskell.org/package/XSaiga, 2020 (cit. on p. 76).

[2] S. M. Peelar and R. A. Frost. "A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore". In: *IEEE 14th International Conference on Semantic Computing, ICSC 2020, San Diego, CA, USA, February 3-5, 2020*. IEEE, 2020, pp. 257–262. URL: `https://doi.org/10.1109/ICSC.2020.00054` (cit. on p. 61).

[3] S. Peelar. "Accommodating prepositional phrases in a highly modular natural language query interface to semantic web triplestores using a novel event-based denotational semantics for English and a set of functional parser combinators". MA thesis. University of Windsor (Canada), 2016 (cit. on pp. 68, 69, 71–74).

[4] R. A. Frost. *An Event-based Approach for Querying Graph-Structured Data Using Natural Language*. `http://cs.uwindsor.ca/~richard/semantics_presentations/talk_for_GraphQ.ppt`. [Online; accessed 14-November-2016]. 2014 (cit. on pp. 68, 74).

[5] R. A. Frost, J. Donais, E. Matthews, and W. Agboola. "A Demonstration of a Natural Language Query Interface to an Event-Based Semantic Web Triplestore". In: *ESWC*. Springer LNCS Volume 8798. 2014, pp. 343–348 (cit. on pp. 68, 69).

[6] R. Verborgh, M. Vander Sande, P. Colpaert, S. Coppens, E. Mannens, and R. Van de Walle. "Web-Scale Querying through Linked Data Fragments." In: *LDOW*. Citeseer. 2014 (cit. on p. 75).

[7] S. Ferré. "SQUALL: a controlled natural language as expressive as SPARQL 1.1". In: *International Conference on Application of Natural Language to Information Systems*. Springer. 2013, pp. 114–125 (cit. on p. 65).

[8] R. A. Frost, B. S. Amour, and R. Fortier. "An Event Based Denotational Semantics for Natural Language Queries to Data Represented in Triple Stores". In: *ICSC, 2013 IEEE Seventh International Conference on Semantic Computing*. IEEE. 2013, pp. 142–145 (cit. on p. 72).

[9] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. "YAGO2: A spatially and Temporally Enhanced Knowledge Base from Wikipedia". In: *Artificial intell.* 194 (2013), pp. 28–61 (cit. on p. 64).

[10] S. Ferre. "SQUALL: A Controlled Natural Language for Querying and Updating RDF Graphs". In: *proc. of CNL 2012*. LNCS 7427. 2012, pp. 11–25 (cit. on p. 65).

[11] M. Wendt, M. Gerlach, and H. Düwiger. "Linguistic Modeling of Linked Open Data for Question Answering". In: *proc. of interact. with Linked Data (ILD 2012)[37]* (2012), pp. 75–86 (cit. on p. 64).

[12] L. Champollion. "Quantification in Event Semantics". In: *Talk given at the 6th int. symp. of cogn., Logic and commun.* 2010 (cit. on p. 77).

[13] R. Hafiz and R. Frost. "Lazy Combinators for Executable Specifications of General Attribute Grammars". In: *proc. of the 12th int. symp. on Practical*

Aspects of Declarative lang. (PADL). ACM-SIGPLAN. 2010-01, pp. 167–182 (cit. on pp. 69, 76).

[14] J. Van Eijck and C. Unger. *Computational semantics with functional programming*. Cambridge University Press, 2010 (cit. on p. 65).

[15] R. A. Frost, R. Hafiz, and P. Callaghan. "Parser Combinators for Ambiguous Left-Recursive Grammars". In: *Practical Aspects of Declarative Languages, 10th International Symposium, PADL 2008, San Francisco, CA, USA, January 7-8, 2008*. Ed. by P. Hudak and D. S. Warren. Vol. 4902. Lecture Notes in Computer Science. Springer, 2008, pp. 167–181. URL: https://doi.org/10.1007/978-3-540-77442-6%5C_12 (cit. on p. 76).

[16] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. "DBpedia: A Nucleus for a Web of Open Data". In: *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*. ISWC'07/ASWC'07. Busan, Korea: Springer-Verlag, 2007, pp. 722–735. URL: http://dl.acm.org/citation.cfm?id=1785162.1785216 (cit. on p. 76).

[17] P. Cimiano, P. Haase, and J. Heizmann. "Porting Natural Language Interfaces Between Domains: an Experimental User Study With the ORAKEL System". In: *Proc. 12th Intnl. Conf. on intell. User Interfaces*. ACM. 2007, pp. 180–189 (cit. on p. 63).

[18] R. A. Frost. "Realization of natural language interfaces using lazy functional programming". In: *ACM Computing Surveys (CSUR)* 38.4 (2006), p. 11 (cit. on p. 69).

[19] K. Ahn, J. Bos, D. Kor, M. Nissim, B. L. Webber, and J. R. Curran. "Question Answering with QED at TREC 2005." In: *TREC*. 2005 (cit. on p. 65).

[20] P. Blackburn and J. Bos. "Representation and inference for natural language". In: *A first course in computational semantics. CSLI* (2005) (cit. on p. 65).

[21] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. "Named Graphs, Provenance and Trust". In: *proc. of the 14th int. conf. on World Wide Web*. ACM. 2005, pp. 613–622 (cit. on p. 64).

[22] R. Frost and P. Boulos. "An efficient compositional semantics for natural-language database queries with arbitrarily-nested quantification and negation". In: *Advances in Artificial Intelligence* (2002), pp. 252–267 (cit. on p. 67).

[23] T. Parsons. *Events in the Semantics of English*. Vol. 5. Cambridge, Ma: MIT Press, 1990 (cit. on p. 64).

[24] R. Frost and J. Launchbury. "Constructing natural language interpreters in a lazy functional language". In: *The Computer Journal* 32.2 (1989), pp. 108–121 (cit. on pp. 67, 69).

[25] D. Dowty, R. Wall, and S. Peters. *Introduction to Montague Semantics*. Dordrecht, Boston, Lancaster, Tokyo: D. Reidel Publishing Company, 1981 (cit. on pp. 62, 65, 69).

Chapter 6

A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores

This paper was published as:

S. M. Peelar and R. A. Frost. "A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores". In: *Web Information Systems and Technologies - 15th International Conference, WEBIST 2019, Vienna, Austria, September 18-20, 2019, Revised Selected Papers*. Ed. by A. Bozzon, F. J. D. Mayo, and J. Filipe. Vol. 399. Lecture Notes in Business Information Processing. Springer, 2019, pp. 168–194. URL: https://doi.org/10.1007/978-3-030-61750-9%5C_8

This paper has been published in the WEBIST 2019 Springer Book. We were invited to submit this extended paper following our presentation at the WEBIST 2019 conference.

6.1 Introduction

This is an extended version of the paper by Frost and Peelar [2] that was presented at WEBIST 2019 in Vienna, Austria. That paper was selected as one of the best

papers at WEBIST 2019 and the authors were invited to submit an extended version for publication. In this paper we expand upon the compositionality of our NLQI, including the parsing framework and semantic implementations, we introduce a novel method to accommodate superlatives using compositional semantics, and we discuss a novel approach to memoization and triplestore retrieval. We also significantly expand upon how our NLQI is implemented.

We begin by describing a Natural Language Query Interface (NLQI) that we have built. We hope that the interface will motivate readers to look into our modifications to MS. In Section 6.2, we explain how our NLQI can be accessed through the Web. In Section 6.3, we describe the compositional aspects of our NLQI. In Section 6.4, we describe the Semantic Web triplestore. In Section 6.5 we discuss example queries and their results, including examples of what are often referred to as "non-compositional" features of NL that our NLQI can handle. With each of the examples we provide an informal explanation of how the answer is, or could be, computed. In Section 6.6, we describe the new FDBR data structure which is central to our approach. In Section 6.7 and Section 6.8, we describe how our system accommodates chained prepositional phrases with superlatives. In Section 6.9, we describe how to use our approach with relational databases. In Section 6.10, we provide a system overview and implementation details on how our semantics are realized. Section 6.11 discusses how our work fits into the framework of existing work in this area. We close with Section 6.12 and Section 6.13 where we discuss future research directions and our conclusions.

Much of our semantics is based on MS. We differ in these ways:

1. We add events to the basic ontological concepts of entities and truth values.

2. Each event has a number of roles associated with it. Each role has an entity as a value.

3. For efficiency, we use sets of entities rather than characteristic functions of those sets as is the case in MS.

4. We define transitive n-ary verbs in terms of sets of events, each with n roles.

5. We compute FDBRs, the novel data structure presented in this paper, from sets of events and use them in the denotations of transitive verbs and in computing results of queries containing prepositional phrases. Although not referred to as an FDBR, the use of relational images in denotations of verbs was first proposed by Frost and Launchbury in 1989 [23].

We hope that this paper reawakens an interest in Compositional Semantics, in particular for NL query processing.

6.2 How to Access our NLQI

Our NL interface is accessible via the following URL, and is speech enabled for both voice-in and voice-out in browsers that support the Web Speech API:

`http://speechweb2.cs.uwindsor.ca/solarman4/demo_sparql.html`

6.3 Compositionality

Compositionality is a useful property of any system as it facilitates understanding, construction, modification, extension, proof of properties, and reuse in different situations. When building our system, we tried to make it as compositional as possible: a compositional syntax processor is systematically combined with a compositional semantics.

6.3.1 The Compositionality of our Syntactic Processor

Our parser is designed and built using the Haskell programming language, using parser combinators [16]. The approach enables parsers to be constructed as executable specifications of context-free grammars with explicit and implicit left-recursive productions, which is useful for defining grammars for NL. The result of

applying our parser is the set of all parse trees for ambiguous grammars. The trees are represented efficiently using a Tomita-style [24] compact graph in which trees share common components.

In 2008, Frost and Hafiz [16] demonstrated that it is possible to efficiently implement context-free parsing using combinators, with their approach having $O(n^4)$ complexity in the worst case and $O(n^3)$ complexity in the average case.

The following example was featured in Frost and Hafiz [16]. To demonstrate use of our combinators, consider the following ambiguous grammar from Tomita [24]:

```
s    ::= np vp | s pp    np   ::= noun  | det noun | np pp
pp   ::= prep np         vp   ::= verb np
det  ::= "a"   | "the"   noun ::= "i"   | "man" | "park" | "bat"
verb ::= "saw"           prep ::= "in"  | "with"
```

In this grammar, the non-terminal `s` stands for sentence, `np` for nounphrase, `vp` for verbphrase, `det` for determiner, `pp` for prepositional phrase, and `prep` for preposition. It is left recursive in the rules for `s` and `np`. The Haskell code below defines a parser for the above grammar using our combinators `term` (terminal), `<+>` (alternative), and `*>` (sequence) [16]:

```
data Label = S | ... | PREP
s    = memoize S    $ np *> vp <+> s *> pp
np   = memoize NP   $ noun <+> det *> noun <+> np *> pp
pp   = memoize PP   $ prep *> np
vp   = memoize VP   $ verb *> np
det  = memoize DET  $ term "a" <+> term "the"
noun = memoize NOUN
$ term "i" <+> term "man" <+> term "park" <+> term "bat"
verb = memoize VERB $ term "saw"
prep = memoize PREP $ term "in" <+> term "with"
```

Parsers written in this fashion are highly compositional, and can be easily extended with new rules if needed. Parsers constructed with our combinators have $O(n^3)$

worst case time complexity for non-left-recursive ambiguous grammars (where n is the length of the input), and $O(n^4)$ for left recursive ambiguous grammars. This compares well with $O(n^3)$ limits on standard algorithms for CFGs such as Earley-style parsers [25]. The increase to n^4 is due to expansion of the left recursive non-terminals in the grammar. The potentially exponential number of parse trees for highly-ambiguous input are represented in polynomial space as in Tomita's algorithm.

6.3.2 The Compositionality of our Semantics

The semantics on which our system is based is similar to Montague Semantics. All phrases of the same syntactic category have meanings of the same semantic type. The meaning of all words and phrases are functions defined over sets of base terms which are entities, events and Boolean values. The meaning of a complex phrase is obtained by applying the functions which are the meanings of its parts, to each other in an order determined by the syntactic structure of the whole. Our system was easy to construct, and is easy to extend. Additional language features are accommodated by adding their syntactic structure and then defining their semantics by viewing the semantics of words and phrases of the same syntactic category.

6.3.3 The Compositionality of the Whole NL Processor

Our processor is built as an executable specification of a fully general attribute grammar. Compositional semantic rules are added to each syntactic production using the technique of Frost, Hafiz and Callaghan [16]. The attribute grammar is fully general as it can accommodate left recursive context-free grammars and fully-general dependencies between inherited and synthesized attributes. Haskell allows any computational dependency between attributes to be defined. Also, Haskell's lazy evaluation strategy enables our language processor to be efficient. For example, no attribute computation is carried out until a successful parse has been obtained. We have also developed a variation of memoization using monads [16]

in order to reduce the complexity of syntactic and semantic evaluation. In the paper by Frost and Peelar [2] we discuss how we accommodate, using our compositional approach, various English phrases that are often given as examples of non-compositional constructs.

6.4 The Triplestore that is Queried

Our NLQI computes answers with respect to an *event-based* Semantic Web triplestore containing data about the planets, the moons that orbit them, and the people who discovered those moons, and when, where and with what implement they were discovered. Briefly, a triplestore is a database of 3-tuples, called triples, that have the form (***subject***, *predicate*, ***object***), where ***subject***, *predicate* and ***object*** are Uniform Resource Identifiers (URIs).

An *event-based triple* has a ***subject*** that identifies an event rather than an entity [5]. In these triples, the *predicate* identifies a *role* through which the ***object*** participates in the event. That is, an event-based triple (e, r, o) expresses that o participates in e through role r. We call o the event e's "*r property*". For example, in Table 4.1, "hall" is event "event1045"'s ***subject*** property. Triplestores consisting of event-based triples are called *event-based triplestores*.

The advantage of event-based triplestores is that additional information about the events and entities participating in those events is immediately available. This is not the case in an entity-based triplestore, where some form of *reification* is necessary to obtain additional information about a fact expressed in a triple. For example, obtaining the location where "`hall discovered phobos`" in an entity-based triplestore, described by $(\mathsf{hall}, \mathsf{discovered}, \mathsf{phobos})$, is not possible without reification.

We assume that each event will at minimum contain a role *ev_ type* that identifies the type of the event, with the general expectation that events of the same type will contain similar roles. This implies the existence of a schema that describes the types of roles that an event may contain. As a consequence of this, each event could be equally well be represented by a row in a relational database. We discuss this

further in Section 6.9.

Going forward, when we refer to the type of an event or set of events, we are referring to their *ev_ type* property. Likewise, when we refer to events of a particular type, we are referring to events whose *ev_ type* property corresponds to that type. As a shorthand, we use *t*-type events to refer to events with type *t*. For example, "*discover*" events refers to events that have *ev_ type* property "*discover*".

The triplestore contains triples such as those in Table 4.1 which represent the event in which *hall* (in the role of "*subject*") discovered *phobos* (in the role of "*object*") in 1877 (in the role of "*year*") with the `refractor_telescope_1` (in the role of "*implement*") at the `us_naval_observatory` (in the role of "*location*"). Events representing set membership are represented as shown in Table 4.2.

The complete triplestore, which contains tens of thousands of triples, is hosted on a remote server using the Virtuoso software [12] and can be accessed by following the link at the beginning of Section 6.2.

6.5 Example Queries

Our NLQI can answer millions of queries with respect to the triplestore discussed above. The NLQI can accommodate queries containing common and proper nouns, adjectives, conjunction and disjunction, intransitive and transitive verbs, nested quantification, superlatives, chained prepositional phrases containing quantifiers, comparatives and polysemantic words. In the following sections, we provide an informal explanation of how the answer is computed. If a query is syntactically ambiguous, the results from each possible interpretation of the query are separated with a semicolon.

6.5.1 Queries Demonstrating the Range of NL Features that our NLQI can Accommodate

`phobos spins` ⇒ *True*

`phobos is a moon` ⇒ *True*

The function denoted by "`phobos`" checks to see if e_{phobos} is a member of the **spin** set, and secondly if e_{phobos} is a member of the **moon** set.

`a moon spins` ⇒ *True*

`every moon spins` ⇒ *True*

`an atmospheric moon exists` ⇒ *True*

The function denoted by "`a`" checks to see if the intersection of the *moon* set and the *spin* set is non-empty. The function denoted by "`every`" checks to see if the *moon* set is a subset of the *spin* set. The denotations of "`a`" and "`every`" that we use are set-theoretic event-based versions of the denotations from MS which use characteristic functions. The answer to the third query is obtained by checking if the intersection of the *atmospheric* set and the *moon* set is non-empty.

`hall discovered` ⇒ *True*

All of the events of type "***discover***" are collected together and are checked to see if e_{hall} is found as the subject role value of any of them. If so, *True* is returned.

`when did hall discover` ⇒ *1877*

The *year* property of the events returned by "`hall discover`" (treated as "`hall discovered`") are returned.

`phobos was discovered` ⇒ *True*

All of the events of type "***discover***" are collected together and are checked to see if e_{phobos} is found as the ***object*** role value of any of them. If so *True* is returned.

`earth was discovered` ⇒ *False*

Earth was not discovered by anyone, according to our data.

`did hall discover phobos` ⇒ *True*

All of the events of type "***discover***" are collected together and are checked to create a pair (s, evs) for each value of the ***subject*** property found in the set of events. *evs*

is the set of events to which the ***subject*** property is related through a discovery event. Each pair is then examined to see if the function denoted by the object termphrase (in this case "`phobos`") returns a non-empty set when applied to a set (called an *FDBR*, which is described in Section 6.6) generated from the set of ***evs*** in the pair, and if so the subject of the pair is added to the set which is returned as the denotation of the verbphrase part of the query. The denotation of the termphrase at the beginning of the query is then applied to the denotation of the verbphrase to obtain the answer to the query.

Owing to the fact that our semantics is compositional, the ***subject*** and ***object*** termphrases of the query above can be replaced by any termphrases, e.g.:

`a person or a team discovered every moon that orbits mars` ⇒ *True*

`who discovered 2 moons that orbit mars` ⇒ *hall*

"`who`", "`what`", "`where`", "`when`" and "`how`" can be used in place of the ***subject*** termphrase. Different role values are returned depending on which "*wh*"-word is used in the query:

`where discovered by galileo` ⇒ *padua*

`when discovered by galileo` ⇒ *1610*

`every telescope was used to discover a moon` ⇒ *True* (w.r.t.our data)

`a moon was discovered by every telescope` ⇒ *False*

`a telescope was used by hall to discover two moons` ⇒ *True*

`which moons were discovered with two telescopes`
⇒ *halimede laomedeia sao themisto*

`who discovered deimos with a telescope that was used to discover every moon that orbits mars` ⇒ *hall*

`who discovered a moon with two telescopes`
⇒ *nicholson science_team_18 science_team_2*

`how was sao discovered` ⇒ *blanco_telescope canada-france-hawaii_telescope*

`how discovered in 1877` ⇒ *refractor_telescope_1*

`how many telescopes were used to discover sao` ⇒ *2*

`who discovered sao` ⇒ *science_ team_ 18*

`how did science_team_18 discover sao`

⇒ *blanco_ telescope canada-france-hawaii_ telescope*

`which planet is orbited by every moon that was discovered by two people` ⇒ *saturn; none* (ambiguous because "`by two people`" could apply to "`discovered`" or "`orbited`")

`which person discovered a moon in 1877 with every telescope that was used to discover phobos` ⇒ *hall; none* (ambiguous because "`to discover phobos`" could apply to "`used`" or "`discovered`")

`who discovered in 1948 and 1949 with a telescope` ⇒ *kuiper*

6.5.2 Queries with "Non-Compositional" Structures

We agree that natural language has non-compositional features but believe that the non-compositionality is mostly problematic when the objective is to give a meaning to an arbitrary NL expression (i.e. an NL expression without a context). It is less problematic when answering NL queries. As illustrated below, the person posing the query, or the database or triplestore can provide contexts that help resolve much of the ambiguity resulting from non-compositional features. The advantages of a using a compositional semantics include:

1. The answer to a query is as correct as the data from which it is derived,
2. The meaning of sub phrases within a query can be discussed formally,
3. The query language can be extended such that all existing phrases maintain their original meanings,
4. The definition of syntax and semantics in the compositional semantics can be used as a blueprint for the implementation of the query processor.

Some researchers have provided examples of what they claim to be non-compositional structures in NL. For example, Hirst [21] gives the example of the verb "depart" which he states is not compositional because its meaning changes with the

prepositional phrase(s) which follow it, and that the definition of compositionality needs to be modified to include the requirement that the function used to compose the meaning of parts must be systematic. We claim that our semantics for verbs is systematic as the denotations of subject and object termphrases, and the possibly empty list of prepositional phrases following the verb, are treated equally and are all used in the same way to filter the set of events of the type associated with the verb, before that set is returned as the denotation of the verb phrase. This is illustrated in the following queries:

`who discovered` ⇒ *bernard bond cassini cassini_ imaging_ science_ team christy dollfus galileo etc...*

No ***subject***, ***object*** or prepositional phrase is given in the query, and so all events of type "***discover***" are returned by the verbphrase and the denotation of the word "`who`" picks out the ***subject***s from those events.

`where discovered io` ⇒ *padua*

No ***subject***, or prepositional phrase is given in the query, and so all events of type "***discover***" are considered and filtered by the denotation of the ***object*** termphrase "`io`" and then, those that pass the filter are returned by the verbphrase and the denotation of the word "`where`" picks out the location from those events.

`who discovered in 1610` ⇒ *galileo*

No ***subject*** or ***object*** is in the query so all events of type "***discover***" are considered and only those with the ***year*** property equal to 1610 pass the filter and then the denotation of the word "`who`" selects the subject which is returned.

`who discovered every moon that orbits mars with one telescope or a moon that orbits jupiter with a telescope` ⇒ *one. ; none. ; none. ; bernard galileo kowal melotte nicholson perrine science_ team_ 1 science_ team_ 2 ; hall ; hall ; none.*

As shown above, in our semantics, the ***subject*** and ***object*** termphrases are treated as filters, as are all prepositional phrases. Note that several results are returned here

because the query is syntactically ambiguous. We discuss solutions on how to best present the results of ambiguous queries to the user in Section 6.10.3.

`where discovered in 1610` ⇒ *padua*

`how discovered in padua` ⇒ *galilean_ telescope_ 1*

These queries retrieve the *location* and *implement* properties of the events of "`discovered in 1610`" and "`discovered in padua`" respectively.

6.5.3 Extensions to the Semantics

Some phrases containing nested quantifiers are given by some researchers as examples of non-compositionality. For example: "a US diplomat was sent to every capital" is often read as having two meanings which can only be disambiguated by additional knowledge. We argue that the person posing a query can express the query unambiguously if they are familiar with quantifier scoping conventions used by our processor, as illustrated in the following:

`christy or science_team_19 or science_team_20 or science_team_21 discovered every moon that orbits pluto` ⇒ *False*

In our semantics, quantifier scoping is always leftmost/outermost, and an unambiguous query can be formulated as follows:

`every moon that orbits pluto was discovered by christy or science_team_19 or science_team_20 or science_team_21` ⇒ *True*

Some examples of non-compositionality involve polysemantic superlative words such as "most" in, for example:

"Who discovered most moons that orbit P. Where P is a planet."

If "`most`" is treated as "more than half" then:

`who discovered most moons that orbit mars` ⇒ *hall*

However, consider the answer to the alternate reading "who discovered the most moons that orbit *P*" – i.e. more than anyone else who discovered a moon that orbits *P*.:

`what discovered the most moons that orbit jupiter` ⇒ *science_team_4*

Here, the *subjects* of the "***discover***" events are sorted based on the cardinality of the number of things they discovered after filtering the events for objects which are moons that orbit *jupiter*. Of the 50 moons that orbit *jupiter*, *science_team_4* discovered 12 of them.

`how was every moon that orbits saturn discovered` ⇒ *cassini reflector_telescope_1 aerial_telescope_1 refractor_telescope_4 etc...*

It may be surprising that *cassini* is returned in the answer since it is not a *telescope*, but is instead a *spacecraft*. However, since it was used to discover at least one *moon* that orbits *saturn*, it is considered to have fulfilled the ***implement*** role and is encoded as such in the triplestore.

6.6 The FDBR: A Novel Data Structure for Natural Language Queries

6.6.1 Quantifiers and Events

In 2015, Champollion [6] stated that, at that time, it was generally thought by linguists that integration of Montagovian-style compositional semantics and Davidsonian–style event semantics [22, 26] was problematic, particularly with respect to quantifiers. Champollion did not agree with that analysis and presented an integration which he called "quantificational event semantics" which he claimed solved the difficulties of integration by assuming that verbs and their projections denote existential quantifiers over events and that these quantifiers always take lowest possible scope.

In this paper, we borrow much from Montague Semantics (MS), Davidsonian Event Semantics, and Champollion's Quantificational Event Semantics. However, we provide definitions of our denotations in the notation of set theory, which improves computational efficiency and, we believe, simplifies understanding of our

denotations. We also believe that our semantics is intuitive, systematic, and compositional.

6.6.2 Montague Semantics

All quantifiers, such as "`a`", "`every`" and "`more than two`" are treated in MS as functions which take two characteristic functions of sets as arguments and return a Boolean value as result. Our modifications to MS are to use sets of entities instead of predicates/characteristic functions of those sets, and to pair sets of events with each entity; the set of events paired with an entity justify the entity's inclusion in the denotation. For example:

$$\|propernoun\| = \lambda p.\{(e, evs) \mid (e, evs) \in p \ \&\ e = \text{the entity associated with the proper noun}\}$$

$$\|spins\| = \{(\mathrm{e}_{\mathrm{phobos}}, \{\mathrm{ev}_{1360}\}), (\mathrm{e}_{\mathrm{deimos}}, \{\mathrm{ev}_{1332}\}), etc \ldots\}$$

Therefore,

$$\begin{aligned}
&\|phobos\ spins\| \\
\Longrightarrow\ &\|phobos\|\ \|spins\| \\
\Longrightarrow\ &\lambda s.\{(e, evs) \mid (e, evs) \in s \ \&\ e = \mathrm{e}_{\mathrm{phobos}}\}\ \|spins\| \\
\Longrightarrow\ &\{(e, evs \mid (e, evs) \in \|spins\| \ \&\ e = \mathrm{e}_{\mathrm{phobos}}\} \\
\Longrightarrow\ &\{(\mathrm{e}_{\mathrm{phobos}}, \{\mathrm{ev}_{1360}\})\}
\end{aligned}$$

We call this set of pairs of entities and events an *FDBR*, and describe it in more detail in Section 6.6.3. In the following example, we show how the FDBR can be used to denote the quantifier "`a`". The function intersect computes the intersection of two FDBRs based on their entities, keeping the events of the second FDBR and

discarding those of the first in the result.

$$\textsf{intersect} = \lambda ms.\{(e_1, evs_2) \mid (e_1, evs_1) \in m \ \&\ (e_2, evs_2) \in s \ \&\ e_1 = e_2\}$$
$$\|a\| = \textsf{intersect}$$

Therefore,

$$\begin{aligned}
&\|a\ moon\ spins\| \\
\Longrightarrow\ &\|a\|\ \|moon\|\ \|spins\| \\
\Longrightarrow\ &\{(e_1, evs_2) \mid (e_1, evs_1) \in \|moon\| \ \&\ (e_2, evs_2) \in \|spins\| \ \&\ e_1 = e_2\} \\
\Longrightarrow\ &\{(\textsf{e}_{\textsf{phobos}}, \{\textsf{ev}_{1360}\}), (\textsf{e}_{\textsf{deimos}}, \{\textsf{ev}_{1332}\}),\ etc \ldots\}
\end{aligned}$$

We can define the denotations of other quantifiers in terms of intersect as well. For example, consider the denotation of "`every`", where ents m denotes the set of entities that appear in the first column of the FDBR m:

$$\textsf{ents} = \lambda m.\{ent \mid (\exists evs)\ (ent, evs) \in m\}$$
$$\|every\| = \lambda ms. \begin{cases} intersect\ m\ s, & \textsf{ents}\ m \subseteq \textsf{ents}\ s \\ \emptyset, & \text{otherwise} \end{cases}$$

Therefore,

$$\begin{aligned}
&\|every\ moon\ spins\| \\
\Longrightarrow\ &\|every\|\ \|moon\|\ \|spins\| \\
\Longrightarrow\ &\textsf{intersect}\ m\ s \quad (\text{since } \textsf{ents}\ \|moon\| \subseteq \textsf{ents}\ \|spins\|) \\
\Longrightarrow\ &\{(\textsf{e}_{\textsf{phobos}}, \{\textsf{ev}_{1360}\}), (\textsf{e}_{\textsf{deimos}}, \{\textsf{ev}_{1332}\}),\ etc \ldots\}
\end{aligned}$$

Note that the events *evs* paired with the entities returned in the denotation of "`was every moon that orbits saturn discovered`" are a subset of the events of type "*discover*" where the *object* property of those events are moons, since the result of *intersect_fdbr* takes the events of from its second argument. This enables additional

data to be accessed from those events, as illustrated in the last example query in the previous section, where "`how`" retrieves the *implement* property from those events. This allows all "*wh*"-style questions to be handled compositionally, selecting the desired properties from the events as needed.

6.6.3 The FDBR

In order to generate the answer to "`hall discovered every moon that orbits mars`", $\|every\|$ is applied to $\|moon\ that\ orbits\ mars\|$ (i.e. the set of *moons* that orbit *mars*), as first argument, and the set of entities that were discovered by *hall*, as the second argument. Our semantics generates this set from the set of events of type "*discover*" whose the subject property is "*hall*", as discussed below:

Every set of n-ary events (i.e. events with n roles) of a given type, e.g. discovery, defines $n^2 - n$ binary relations. For example, for discovery events:

$$\text{discover}_{rel:subject\to object}\ \text{discover}_{rel:subject\to year}\ \text{discover}_{rel:subject\to implement}\cdots$$
$$\text{discover}_{rel:object\to subject}\ \text{discover}_{rel:object\to year}\ \ \text{discover}_{rel:object\to implement}\cdots$$
$$\text{discover}_{rel:year\to subject}\ \ \text{discover}_{rel:year\to object}\ \ \text{discover}_{rel:year\to implement}\cdots$$

etc... to 20 binary relations for the set of discovery events or an 5-ary discovery relation. For example:

$$\text{discover}_{rel:subject\to object} = \{(\text{ev}_{1045}, \text{e}_{\text{hall}}, \text{e}_{\text{phobos}}), (\text{ev}_{1046}, \text{e}_{\text{hall}}, \text{e}_{\text{deimos}}), etc\ldots\}$$

If we collect all of the values from the range of a relation that are mapped to by each value v from the domain (i.e. the image of v under the relation r) and create the set of all pairs $(v, image\_of\_v)$, we obtain a *Function Defined by the Binary*

Relation r, i.e. the FDBR. For example:

$$\text{FDBR}(\text{discover}_{rel:subject \rightarrow object}) = \left\{(\text{e}_{\text{hall}}, \{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}), (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\}), etc \ldots\right\}$$

It is these functions that are created, and used, by the denotation of the transitive verb associated with the type of the events. For example in calculating the value of $\|who\ discovered\ every\ moon\ that\ orbits\ mars\|$, $\|every\|$ is applied to the set of entities which is the denotation of "`moon that orbits mars`" (i.e $\{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}), (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\}$) and all of the images that are in the second field of the pairs in $\text{FDBR}(\text{discover}_{rel:subject \rightarrow object})$.

For the pair $(\text{e}_{\text{hall}}, \{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}), (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\})$, $\|every\|$ returns the non-empty set $\{(\text{e}_{\text{phobos}}, \{\text{ev}_{1045}\}), (\text{e}_{\text{deimos}}, \{\text{ev}_{1046}\})\}$, and the value in the first field, i.e. e_{hall}, is subsequently returned with the answer to the query.

The various FDBRs are used to answer different types of queries. For example:

`who discovered phobos and deimos` $\Rightarrow$ *hall*

uses $\text{FDBR}(\text{discover}_{rel:subject \rightarrow object})$

`where discovered by galileo` $\Rightarrow$ *padua*

uses $\text{FDBR}(\text{discover}_{rel:location \rightarrow subject})$

`how discovered in 1610 or 1855` $\Rightarrow$ *galilean_ telescope_ 1*

uses $\text{FDBR}(\text{discover}_{rel:implement \rightarrow year})$

6.7 Handling Prepositional Phrases

Prepositional phrases (PPs) such as "`with a telescope`" are treated similarly to the method above, except that the termphrase following the preposition is applied to the set of entities that are extracted from the set of events in the FDBR function, according to the role associated with the preposition. The result is a "filtered" FDBR which is further filtered by subsequent PPs.

6.8 Handling Superlative Phrases

A novel feature of our semantics is that we can directly accommodate superlative phrases such as "`most`" and "`the most`" inside chained prepositional phrases. Here, we take "`most`" to mean "more than half" and "`the most`" to mean "more than anything else". This makes it possible to answer queries such as "`who discovered a moon using the most telescopes`" and "`most planets are orbited by a moon`" with our NLQI.

Superlatives can be placed nearly anywhere a determiner can exist. This makes it possible to nest superlatives inside chained prepositional phrases, a property we believe to be novel in our semantics. For example, consider "`what discovered at the most places using the most telescopes`", where "`the most`" occurs inside both prepositional phrases "`at the most places`" and "`using the most telescopes`". The query is always evaluated in left-to-right order, and results are sorted by each superlative phrase in the order they appear. In this case, the results are first sorted by the number of places, followed by the number of telescopes, both in descending order. First, the denotation for "`most`" (as in "more than half") is defined as follows:

$$\|most\| = \lambda ms. \begin{cases} \text{intersect } m\ s, & |\text{intersect } m\ s| > |s|/2 \\ \emptyset, & \text{otherwise} \end{cases}$$

Providing a denotation for superlative phrases such as "`the most`" is more challenging. To achieve this and maintain compositionality, the superlatives are handled in the denotation for the transitive verbs. First, we introduce the denotation for "`the most`":

$$\|the\ most\| = \lambda m.(\text{GT}, intersect\ m)$$

"`the most`" takes a nounphrase as an argument and returns a pair consisting of the ordering GT (i.e. "greater than"), and a termphrase created using partial application of the *intersect* function. This ordering describes how the results should be sorted

– in this case, in descending order.

The denotation for prepositional phrases is modified to include an ordering as third parameter, which may take on the special value *None* if the prepositional phrase does not contain a superlative phrase within it. However, if it does contain a superlative phrase, the ordering of the prepositional phrase is set to the ordering specified in the denotation of the superlative phrase.

The denotation for transitive verbs is modified such that, at the end of the prepositional phrase evaluation performed previously, where the filtered FDBR is obtained (containing only *relevant* events [5]), the resulting FDBR is passed to a new function, `filter_super`, which handles superlative evaluation. The behavior of this function is as follows. First, if no superlatives are present (i.e. the ordering in the denotation of each prepositional phrase is *None*), nothing more is done, and the behavior of the new denotation is identical to the previous one.

If superlatives are present, however, they are evaluated in the order they appear. For each superlative phrase present in the chain of prepositional phrases, the FDBR is expanded to a new data structure called a *Generalized FDBR* (or *GFDBR*) which is similar to an FDBR, except that instead of having a set of events in its second column, it has an FDBR instead. The GFDBR is formed by taking the set of events in each row of the original FDBR, and expanding them into an FDBR using the role attached in the prepositional phrase. This is used to obtain the cardinality of the number of entities that the subject is related to in that role under the FDBR (called the *object cardinality*). Now, these object cardinalities are used to partition the GFDBR into a set of GFDBRs, where the set with the highest (or lowest) object cardinality is chosen to replace the original GFDBR, depending on the ordering in the denotation of the prepositional phrase (i.e. the ordering denoted by the superlative phrase). For "`the most`", it would be the set with the highest object cardinality (since the ordering is GT). In the future, for "`the least`", it would be the set with the lowest object cardinality. The GFDBR is then converted back into an FDBR by keeping only the events in each row, and the process repeats until no more superlative phrases are remaining. The final FDBR is returned as the result.

This allows superlative phrases to still be handled in left-to-right evaluation order, and it also allows results to be sorted by multiple columns. For example "`who discovered the most moons in the most places`" would first sort by "`the most moons`", and following that, would sort by "`the most places`". Currently, we are not able to accommodate "`the least`", as the semantics filters out rows with empty sets of events in FDBRs before superlatives work on them. For example, if a user were to ask "`which planet has the least moons`", the answer currently would be "`earth`", as it has only one moon, and our system filters out both "`venus`" and "`mercury`" (which have no moons) before they have a chance to affect the result. This seems to be related to our original Open World Assumption, where we only include results in the result set if there is at least one accompanying event in the FDBR to justify its inclusion. It is possible that if negation could be accommodated in the semantics, then "`the least`" could be handled as well, since they seem to be related problems.

6.9 Our Approach with Relational Databases

Our NLQI can be easily adapted for use with conventional relational databases. First, note that each event at minimum contains a role *ev_ type* that identifies the type of event, and as noted in Section 6.4, there is a general expectation that events of the same type should contain similar roles. Second, note that the event identifier in each triple is a URI and is therefore unique by definition.

Assume the roles that events of a particular type t are fixed, including optional roles. Let N be the number of roles, including optional roles, that an event of type t contains. Then an event of type t can be described as a row in a relation with N columns, each role occupying one column respectively, with optional roles taking on a special value `NULL` if they are not present in that particular event. Let this relation be called *ev_ type*.

Store this relation in a relational database as a table using the event identifier as the primary key. Now, only the triple retrieval functions in Section 6.10.2 need

to be modified to use this database in place of a triplestore. This architecture allows the denotations to remain unchanged and yet still work with different types of databases. Note that triplestores do have an advantage in that they need not be rebuilt if a new role is added to the event. The decision to choose one approach over the other needs to be weighed based on application specific factors.

6.10 Implementation of our NLQI

We built our query processor as an executable attribute grammar using the *X-SAIGA* Haskell parser-combinator library package [1]. The collect function which converts a binary relation to an FDBR is one of the most compute intensive parts of our implementation of the semantics. However, in Haskell, once a value is computed, it can be made available for future use. We have developed an algorithm to compute FDBR(*rel*) in $O(n \lg n)$ time, where n is the number of pairs in *rel*. Alternatively, the FDBR functions can be computed and stored in a cache when the NLQI is offline. Our implementation is amenable to running on low power devices, enabling it for use with the Internet of Things. A version of our query processor exists that can run on a common consumer network router as a proof of concept for this application. The use of Haskell for the implementation of our NLQI has many advantages, including:

1. Haskell's "lazy" evaluation strategy only computes values when they are required, enabling parser combinator libraries to be built that can handle highly ambiguous left-recursive grammars in polynomial time.
2. The higher-order functional capability of Haskell allows the direct definition of higher-order functions that are the denotations of some English words and phrases.
3. The ability to partially apply functions of n arguments to 1 to n arguments allows the definition and manipulation of denotation of phrases such as "`every moon`", and "`discover phobos`".
4. The availability of the *hsparql* [15] Haskell package enables a simple interface

between our semantic processor and SPARQL endpoints to our triplestores.

6.10.1 System Architecture

A flowchart of our system architecture is presented in Section 6.10.1.

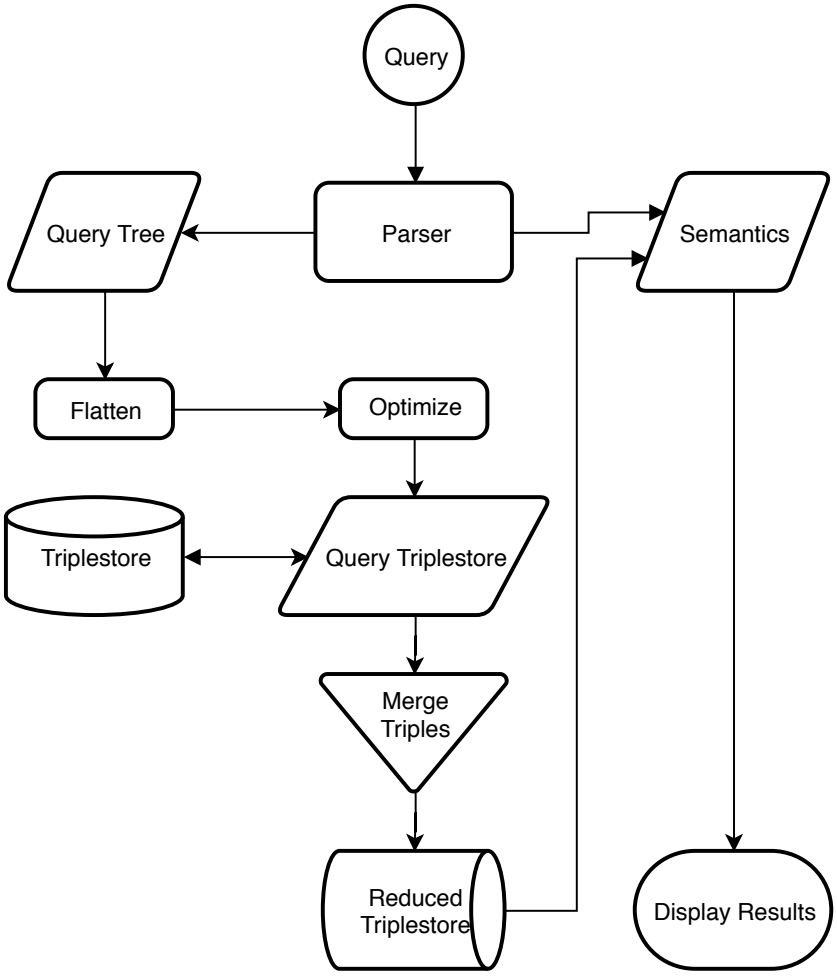

Figure 6.1: Application architecture.

The query begins as a string of text as sent to the semantics, which is then sent directly to the parser, as described in Section 6.3.1. This produces two results:

(1) A function that, given a set of triples, will evaluate the query with respect to that set of triples and return the result

(2) A "*Memo Tree*" that roughly follows the syntax tree resulting from the parse of the input string. In addition to providing a unique name to each sub-

expression of the parsed input, it is also used to determine which queries need to be evaluated against the remote triplestore.

The function produced in (1) requires a set of triples to produce a result. While it is possible, given sufficient time and resources, to directly retrieve all triples from the remote triplestore and pass them directly into this function to evaluate the input, in practice it is cost prohibitive to do so.

Instead, we retrieve only *relevant triples* [5] from the remote triplestore and we create a *reduced* triplestore from them which is then passed into (1). The Memo Tree obtained in (2) is traversed to obtain the set of all triplestore queries that are required to evaluate each sub-expression of the parsed input. These queries correspond to the `getts` family of functions described in Section 6.10.2. The results of these queries may *overlap*, i.e. share triples in common with those of other queries in the set. An optimization step is performed to eliminate these redundant queries. Domain specific knowledge could be used to improve this process where appropriate. Finally, these optimized queries are evaluated against the remote triplestore and the results are merged and stored locally in the reduced triplestore. These triples are then passed to the function produced in (1), yielding the final result. This is one area where our NLQI differs from other NLQIs to the Semantic Web – notice that nowhere do we attempt to directly translate the NL query into SPARQL or any other querying language. Instead, we rely on simple triple querying primitives which are embedded in the semantics to perform this task for us.

The architecture presented in this section lends itself to a very clean implementation in Haskell, where the semantics themselves can be written as pure functions, with the only impure parts of the NLQI being those that directly deal with querying the triplestore and with presenting these results to the user. We expand on the individual sub-components of the NLQI in the following sections.

6.10.2 Triple Retrieval

Remote Triplestore

Our semantics does not directly depend upon any particular query language. When querying remote triplestores, the NLQI requires only two conceptually simple functions. The first is:

```
getts_triples_entevprop_type ev_data prop_names ev_type
```

This function is used to retrieve triples belonging to the relation `ev_type`. `prop_names` is a list of columns of the relation to retrieve. Only the names of the columns of the relation that are actually required are listed here. Finally, `ev_data` is the URL used to access the remote triplestore or database. For example, in the query "`what discovered`", it may be invoked as follows:

```
getts_triples_entevprop_type url ["subject"] "discover_ev"
```

This would retrieve the triples of all "*discover*" events that contain a *subject* property, including the triples describing the type of those events. The second function is:

```
getts_triples_members ev_data set
```

Here, `ev_data` performs the same function as it did previously, and "`set`" indicates the name of a set, for example the moons or the set of things that spin. This retrieves the triples of all "membership" events whose *object* property corresponds to that set, including the triples describing the type of those events.

Together, these two primitives can be used to retrieve triples from event-based triplestores, provided the names of the roles to be queried are known. This would typically be described in a schema, but in simple cases may be feasible to hard-code into a program. To see how these two primitives work in action, consider the following complex query, featuring chained prepositional phrases:

```
which person discovered a moon in 1877 with a telescope
```

This would invoke the following queries to the database:

```
getts_triples_entevprop_type url ["subject", "object", "year",
```

```
"implement"] "discover_ev"
   getts_triples_members url "moon"
   getts_triples_members url "telescope"
   getts_triples_members url "person"
```

These four queries to the remote triplestore, taken together, will retrieve enough information to answer the user's query. Transitive and intransitive verbs are implemented in terms of `getts_triples_entevprop_type`. Common nouns and adjectives are implemented in terms of `getts_triples_members`. These conceptually simple functions are easy to implement in SPARQL, SQL, and as Triple Pattern Fragments [7]. An example implementation is provided in our source code, available on Hackage [1] for both Triple Pattern Fragments and SPARQL.

After all "`getts`" queries are evaluated, their results are merged together into a local *reduced* triplestore. The idea behind this triplestore is that it contains enough triples to evaluate the correct result, but no more than that. In other words, the results from passing in the entire triplestore to the semantic function in (1) and the results from passing in the reduced triplestore should be equivalent.

Reduced Triplestore

Once the reduced triplestore is passed into the semantics, however, it still needs to be queried by the semantic functions in the denotations. This is where the boundary of the impure code of the NLQI meets the pure code of the semantics. At this higher level, there are three primitives that are used to query the reduced triplestore:

- `pure_getts_triples_entevprop_type ev_data prop_names ev_type`
- `pure_getts_triples_entevprop ev_data prop_names evs`
- `pure_getts_members ev_data set`

These are very similar functions to those described previously, however they are implemented as pure functions in Haskell. The actual implementation of the reduced triplestore is opaque to the semantics, which rely strictly on these three functions

to retrieve triples from the reduced triplestore. Implementing these as pure functions allows them to be embedded in the semantics, which are implemented as pure functions themselves. This provides a number of benefits, including allowing the semantics and queries to be lazily evaluated. `pure_getts_triples_entevprop_type` performs a similar role as it did previously. `pure_getts_triples_entevprop` is a new function that, instead of specifying an event type parameter, specifies a set of events instead. This is used to implement chained prepositional phrases, where sets of events are honed down in the order that the phrases occur in (from left to right). Finally, `pure_getts_members` performs a similar function as it did previously, except this time it directly returns an FDBR from the members of the set given to the events in which the set membership is recorded.

6.10.3 Handling Ambiguity in the Query Interface

Syntactic ambiguity

As queries may be ambiguous, it's important that users see how their queries were parsed to understand the result given. Our system displays the parse tree along with the query result to assist with this. The parse tree is presented in a familiar Haskell syntax to indicate scoping. As an example, consider the scoping of the simple query "`who discovered a moon that orbits mars`":

```
who (discovered (a (moon `that` (orbits mars))))
```

Here, we see that scoping of denotations is shown with parentheses. Prepositional phrases are enclosed inside square brackets, with commas to delimit chained prepositional phrases:

```
who discovered a moon in 1877 with a telescope
    ⇒ who (discovered (a moon) [in 1877, with (a telescope)])
```

This mirrors the familiar list syntax that Haskell offers and suggests to the user that the prepositional phrases will be evaluated in the order presented (left to right), allowing users to understand exactly how their query is evaluated by the system.

Now, consider the following ambiguous query:

```
who discovered a moon that orbits in 1877
```

There are two possible parses of this query, depending on which transitive verb the prepositional phrase "`in 1877`" is applied to:

`who (discovered (a (moon `that` (orbits [in 1877]))))` ⇒ *none*

`who (discovered (a (moon `that` orbits)) [in 1877])` ⇒ *hall*

In the first case, the prepositional phrase "`in 1877`" is treated as though it applies to "`orbits`". However, the result is "*none*" because orbit events do not have a concept of time in our database. If we were to add a *year* role to the "orbit" relation, then all planets and moons in the solar system would be returned. In the second case, "`in 1877`" applies to "`discovered`", a relation which has the concept of a time of discovery (the *year* role). As *hall* is the only person that discovered anything in 1877, only they are included in the result.

Our system permits highly ambiguous input, providing a result for each possible parse of that input. However, it may be the case that a user has a clear understanding of how they want their query to be parsed and would gain no benefit from seeing other possible parses of their query. Fortunately, this use case is easily accommodated with a simple extension to our NLQI: allowing the scoping syntax as presented above directly in the query interface itself. For example, a user could directly query "`what (discovered (a (moon `that` orbits)) [in 1877])`", which would exclude the other parse as mentioned in the example above. In fact, the query need not even be fully explicitly scoped to benefit from this. A partial scoping such as "`what discovered (a moon that orbits) in 1877`" would be sufficient to exclude the other undesirable parses from the result. We intend to implement this functionality in our NLQI in the very near future.

It may also be worthwhile to implement a simple dialogue-based approach to disambiguation, where the system could simply provide the possible parses to the user and allow them to choose which one they intended. This approach may be beneficial when using speech to interact with the system, as providing scoping with

the above method directly with speech would be very inconvenient. An example dialogue could be:

User:

```
what discovered a moon that orbits mars in 1877 with a telescope
```

Interface:

```
There are three possible ways I can interpret this query.
Which one do you mean?
1) what (discovered (a (moon `that` (orbits mars [in 1877,
with (a telescope)])))))
2) what (discovered (a (moon `that` (orbits mars))) [in 1877,
with (a telescope)])
3) what (discovered (a (moon `that` (orbits mars [in 1877])))
[with (a telescope)])
```

User:

```
2
```

Interface:

```
OK -- the result of the second interpretation is ``hall''
```

If the modality of the interface is by voice, reading the scoping directly as presented above may be inconvenient to users. Fortunately, it is possible to verbally state the scoping in an intuitive way:

User:

```
what discovered a moon that orbits mars in 1877 with a telescope
```

Interface:

```
I can interpret this three different ways. In the first interpretation,
the prepositions ``in 1877'' and ``with a telescope'' apply to the
verb ``orbit''. Is that what you meant?
```

User:

```
no
```

Interface:

```
   In the second interpretation, the prepositions ``in 1877'' and
``with a telescope'' apply to the verb ``discovered''. Is that
what you meant?
```

User:

```
   yes, that's what i meant
```

Interface:

```
   OK -- the result of that interpretation is ``hall''
```

Given the different nature of the user's responses compared to the queries themselves, they may be subject to a different grammar or may be handled by a different system entirely that permits more free-form responses to be given. This could be a good opportunity to integrate Machine Learning-based Natural Language Processing (NLP) approaches in the NLQI in the future, as they are ideally suited to use cases involving loosely structured input.

Semantic Ambiguity

Semantic ambiguity may also be accommodated by permitting multiple definitions of the same terminal in the grammar, augmenting it with a human readable description of what the terminal means. Each definition would be evaluated as though it were a different parse of the query, although each parse would have the same syntax tree. To avoid confusion, the human readable definition of the word could be printed below the tree.

6.10.4 Semantic Implementation

The semantics themselves are completely unaware of the structure of the underlying triplestore and the methods and query languages used to retrieve triples from it. Recall from Section 6.10.1 that the result of a parse of user input produces two items: a pure function that, given a triplestore as input will produce the result of a query and a tree that represents the query itself, including the types of queries that are required from a remote triplestore.

Applying Multiple Semantics in Parallel

The Biapplicative Bifunctor in Haskell, which is inspired from its counterpart in category theory, can serve as a generalization of function application. One possible use for it is to apply pairs of values to pairs of functions. Briefly, given two arbitrary functions f and g and two values a and b we can use the biapplicative operator `<<*>>` to apply a and b both functions in parallel: (f,g) `<<*>>` $(a,b) = (f\ a, g\ b)$. The functions themselves need not be related. First, we introduce an operator, `>|<`, that allows us to bridge together two semantics such that they can be applied using `<<*>>`:

$$a \texttt{ >|< } b = (a,b)$$

This allows these two independent functions to be applied in parallel while parsing the input string using the exact same grammar and no code duplication, provided the `<<*>>` is used in place of function application. For example, "`a moon spins`" is evaluated as though it were written as "`a <<*>> moon <<*>> spins`" under this approach. Our NLQI uses this to construct the Memo Tree in parallel while applying the denotations of the words in the query. Consider the following example, where GIntersect and GMembers are constructors of the Memo Tree:

```
a' = a >|< GIntersect
moon' = moon >|< GMembers "moon"
spins' = spins >|< GMembers "spins"
```

Therefore,

```
a' <<*>> moon' <<*>> spins'
⟹ (a moon spins, GIntersect (GMembers "moon") (GMembers "spin"))
```

However, this is somewhat inconvenient and unfamiliar syntax to work with. Fortunately, it is trivial to define a set of "wrapper" functions to restore the original

function application syntax:

$$wrap_N\ (f,g)\ (a_1,b_1)\ (a_2,b_2)\ \ldots\ (a_N,b_N) = (f\ a_1\ a_2 \ldots a_N,\ g\ b_1\ b_2 \ldots b_N)$$

Here, the function $wrap_N$ takes a pair of functions (f,g) with arity N and then N pairs of arguments to be applied in order to f and g respectively. This allows "`a' <<*>> moon' <<*>> spins'`" above to be written as "$\mathtt{a}'$ `moon spins`", where $\mathtt{a}'' = wrap_2\ \mathtt{a}'$. Therefore, we can retain the familiar function application syntax in the semantics while taking advantage of parallel function application. By itself, this is a convenience, but let us revisit the Memo Tree once more. It has two uses. The first is as stated previously, in determining which queries need to be performed against the remote triplestore. The second is that this allows us to assign a unique identifier to each sub-expression of the parsed input.

Memoized Compositional Semantics

Consider the query "`what is orbited by a thing that was discovered by a person that discovered phobos`", containing three nested transitive verbs. One possible parse of this query yields:

```
what (is orbited [by (a (thing `that` (was discovered [by (a (person
                `that` (discovered phobos))])))])
```

A query's sub-expressions may be evaluated multiple times during the prepositional filtering of a transitive verb (i.e one evaluation for each row of the FDBR denoted in that transitive verb). This has a compounding effect when transitive verbs are nested as sub-expressions in prepositional phrases of other transitive verbs. In general, if there are m nested transitive verbs in a query, each having an FDBR with n rows. then the complexity for evaluation is $O(n^m)$.

As it turns out, we can use the Memo Tree to memoize the results of the sub-expressions of a query, drastically reducing the number of re-evaluations performed. The memoization occurs in a more sophisticated version of the $wrap_N$ functions described previously, which use the unique identifier provided by the Memo Tree

to memoize the results of the semantic functions as they are evaluated. This is completely transparent to the user, and the familiar function application syntax used in all previous examples still remains. This reduces the complexity to $O(mn)$, where m is the number of nested transitive verbs, each having an FDBR with n rows. All sub-expressions in the query are memoized, including the final result of the query expression itself.

The State monad in Haskell is used to thread the memoized state throughout the execution of the semantics. This mirrors the memoization technique used in the parser itself to provide efficient parsing using combinators [16]. We believe this two-pronged approach to triplestore retrieval and memoization is novel and has not been used in any other Compositional Semantics-based systems. We intend to expand more on our approach in a future publication, as we believe it to be useful for creating modular and efficient compositional NLQIs that can scale to the needs of the Semantic Web. For example, this approach could be used for developing NLQIs for low-power embedded devices that are suitable for IoT applications.

6.11 Related Work

Orakel [18] is a portable NLQI which uses a Montague-like grammar and a lambda calculus semantics. Our approach is similar in this respect. Queries are translated to an expression of first order logic enriched with predicates for query and numerical operators. These expressions are translated to SPARQL or F-Logic. Orakel supports negation, limited quantification, and simple prepositional phrases.

YAGO2 [9] is a semantic knowledge base containing reified triples extracted from Wikipedia, WordNet and GeoNames, representing nearly 0.5 billion facts. Reification is achieved by tagging each triple with an identifier. However, this is hidden from the user who views the knowledge base as a set of "SPOTL" quintuples, where T is for time and L for location. The SPOTLX query language is used to access YAGO2. SPOTLX can handle queries with prepositional aspects involving time and location. However, no mention is made of chained complex PPs.

Alexandria [11] is an event-based triplestore, with 160 million triples (representing 13 million n-ary relationships), derived from FreeBase. Alexandria uses a neo-Davidsonian [22] event-based semantics. In Alexandria, queries are parsed to a syntactic dependency graph, mapped to a semantic description, and translated to SPARQL queries containing named graphs. Queries with simple PPs are accommodated. However, no mention is made of negation, nested quantification, or chained complex PPs.

The systems referred to above have made substantial progress in handling ambiguity and matching NL query words to URIs. However, they appear to have hit a roadblock with respect to natural-language coverage. Most can handle simple PPs such as in "`who was born in 1918`" but none can handle chained complex PPs, containing quantifiers, such as "`in us_naval_observatory in 1877 or 1860`".

Blackburn and Bos [20] implemented lambda calculus with respect to natural language, in Prolog, and Van Eijck and Unger [13] have extensively and clearly discussed such implementation in Haskell. Implementation of the lambda calculus for open-domain question answering has been investigated by [19]. The SQUALL query language [8, 10] is a controlled natural language (CNL) for querying and updating triplestores represented as RDF graphs. SQUALL can return answers directly from remote triplestores, as we do, using simple SPARQL-endpoint triple retrieval commands. It can also be translated to SPARQL queries which can be processed by SPARQL endpoints for faster computation of answers. SQUALL can handle quantification, aggregation, some forms of negation, and simple unchained prepositional phrases containing the word "`at`" and "`in`". It can also handle superlative phrases as long as they are not nested under a prepositional phrase. Notably, the scope of prepositional phrases in SQUALL are the entire sentence they reside in. It is also written in a functional language. However, some queries in SQUALL require the use of variables and low-level relational algebraic operators (see for example, the queries on page 118 of [8]).

6.12 Future Work

Negation

Our system currently relies on the Open World Assumption, where the absence of evidence cannot be treated as having evidence of absence. As a consequence of this, the system currently is unable to handle negation, and does not have a denotation for the words "`no`" and "`not`".

However, there is a clear need for handling negation in our semantics where the Closed World Assumption holds. For example, it should be possible to answer queries such as "`who did not discover a moon`" or "`what discovered no moon`". Work has been done on event-based semantics that can handle negation [6]. We believe it should be possible to accommodate negation in our semantics as well using a similar approach, and in turn provide a denotation for "`the least`" as well, as noted in Section 6.8.

DBPedia

With the addition of memoization in our semantics, we feel our approach is now scalable enough to work directly with DBPedia. We intend to expand on how our semantics can handle large triplestores such as DBPedia in a future publication. In particular, an interface to DBPedia will allow our approach to be directly evaluated with existing systems in use, such as YAGO [9].

Hardware Acceleration

Consider that the reduced triplestore described in Section 6.10.2 is stored locally in the query interface and is queried with the pure "`getts`" functions. These could make good candidates for offloading to FPGA fabric or a GPU for hardware acceleration. Work has been done in developing on FPGAs using Haskell [14]. This could allow for both low latency and low power consumption in embedded consumer devices, such as those that operate on the Internet of Things.

Non-Event-Based Triplestores

We also believe it should be possible to handle non-event based triplestores as well using our approach using a translation layer. It may be possible to use ontological information to provide an event-based view to many kinds of non-event based data. Machine Learning approaches could provide a way forward in the absence of or lacking sufficient ontological information about a triplestore.

6.13 Conclusions

This work comes at an appropriate time when massive triplestores, such as DBpedia [17] are being created containing billions of verified facts. We are currently looking at how such facts can be converted to event-based triples which can be queried by our interface. We are confident that, after we accommodate negation, our compositional semantics is appropriate for answering most queries that are likely to be asked of data stores containing domain-specific knowledge. We have shown how the FDBR data structure presented in this paper can be used to handle many kinds of complex language features, including chained prepositional phrases and superlatives. The way quantification is handled within the semantics is consistent with other work in this area, as discussed in Section 6.6.1. Our approach is extensible enough that it can accommodate queries to both relational and non-relational types of database, including Semantic Web triplestores. Our approach is also suitable for use on low power devices, which may be useful for applications on the Internet of Things (IoT).

We have shown how our system is tolerant of highly ambiguous user input and we discussed possible ways to present this in Section 6.10.3. In particular, we discussed how both semantic and syntactic ambiguity could be handled. We also presented a novel approach to memoizing compositional semantics using unique identifiers attached to sub-expressions in a query, substantially improving the time complexity of evaluation. We also showed how those unique identifiers are also useful to determine the set of queries that need to be made to the remote database.

Our next goal is to provide an NLQI to DBPedia using our approach with the

techniques described here, and then evaluate the effectiveness of our system relative to other NLQIs using established benchmarks, such as QALD [4].

Bibliography

[1] R. Hafiz, R. Frost, S. Peelar, P. Callaghan, and E. Matthews. *The XSAIGA Package*. https://hackage.haskell.org/package/XSaiga, 2020 (cit. on pp. 102, 106).

[2] R. A. Frost and S. M. Peelar. "A New Data Structure for Processing Natural Language Database Queries". In: *Proceedings of the 15th International Conference on Web Information Systems and Technologies, WEBIST 2019, Vienna, Austria, September 18-20, 2019*. 2019, pp. 80–87. URL: `https://doi.org/10.5220/0008124300800087` (cit. on pp. 82, 87).

[3] S. M. Peelar and R. A. Frost. "A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores". In: *Web Information Systems and Technologies - 15th International Conference, WEBIST 2019, Vienna, Austria, September 18-20, 2019, Revised Selected Papers*. Ed. by A. Bozzon, F. J. D. Mayo, and J. Filipe. Vol. 399. Lecture Notes in Business Information Processing. Springer, 2019, pp. 168–194. URL: `https://doi.org/10.1007/978-3-030-61750-9%5C_8` (cit. on p. 82).

[4] R. Usbeck, R. H. Gusmita, A.-C. N. Ngomo, and M. Saleem. "9th challenge on question answering over linked data (QALD-9)". In: *Semdeep/NLIWoD@ ISWC*. 2018, pp. 58–64 (cit. on p. 117).

[5] S. Peelar. "Accommodating prepositional phrases in a highly modular natural language query interface to semantic web triplestores using a novel event-based denotational semantics for English and a set of functional parser combinators". MA thesis. University of Windsor (Canada), 2016 (cit. on pp. 87, 100, 104).

[6] L. Champollion. "The interaction of compositional semantics and event semantics". In: *Linguistics and Philosophy* 38.1 (2015), pp. 31–66 (cit. on pp. 94, 115).

[7] R. Verborgh, M. Vander Sande, P. Colpaert, S. Coppens, E. Mannens, and R. Van de Walle. "Web-Scale Querying through Linked Data Fragments." In: *LDOW*. Citeseer. 2014 (cit. on p. 106).

[8] S. Ferré. "SQUALL: a controlled natural language as expressive as SPARQL 1.1". In: *International Conference on Application of Natural Language to Information Systems*. Springer. 2013, pp. 114–125 (cit. on p. 114).

[9] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. "YAGO2: A spatially and Temporally Enhanced Knowledge Base from Wikipedia". In: *Artificial intell.* 194 (2013), pp. 28–61 (cit. on pp. 113, 115).

[10] S. Ferre. "SQUALL: A Controlled Natural Language for Querying and Updating RDF Graphs". In: *proc. of CNL 2012*. LNCS 7427. 2012, pp. 11–25 (cit. on p. 114).

[11] M. Wendt, M. Gerlach, and H. Düwiger. "Linguistic Modeling of Linked Open Data for Question Answering". In: *proc. of interact. with Linked Data (ILD 2012)[37]* (2012), pp. 75–86 (cit. on p. 114).

[12] O. Erling and I. Mikhailov. "Virtuoso: RDF support in a native RDBMS". In: *Semantic Web Information Management*. Springer, 2010, pp. 501–519 (cit. on p. 88).

[13] J. Van Eijck and C. Unger. *Computational semantics with functional programming*. Cambridge University Press, 2010 (cit. on p. 114).

[14] C. Baaij. "Cλash: From haskell to hardware". MA thesis. University of Twente, 2009 (cit. on p. 115).

[15] J. Wheeler. "The hsparql Package". In: *The Haskell Hackage Repository*. http://hackage.haskell.org/package/hsparql-0.1.2. 2009 (cit. on p. 102).

[16] R. A. Frost, R. Hafiz, and P. Callaghan. "Parser Combinators for Ambiguous Left-Recursive Grammars". In: *Practical Aspects of Declarative Languages, 10th International Symposium, PADL 2008, San Francisco, CA, USA, January 7-8, 2008*. Ed. by P. Hudak and D. S. Warren. Vol. 4902. Lecture Notes in Computer Science. Springer, 2008, pp. 167–181. URL: https://doi.org/10.1007/978-3-540-77442-6%5C_12 (cit. on pp. 84–86, 113).

[17] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. "DBpedia: A Nucleus for a Web of Open Data". In: *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*. ISWC'07/ASWC'07. Busan, Korea: Springer-Verlag, 2007, pp. 722–735. URL: http://dl.acm.org/citation.cfm?id=1785162.1785216 (cit. on p. 116).

[18] P. Cimiano, P. Haase, J. Heizmann, and M. Mantel. *Orakel: A portable natural language interface to knowledge bases*. Tech. rep. Technical report, Institute AIFB, University of Karlsruhe, 2007 (cit. on p. 113).

[19] K. Ahn, J. Bos, D. Kor, M. Nissim, B. L. Webber, and J. R. Curran. "Question Answering with QED at TREC 2005." In: *TREC*. 2005 (cit. on p. 114).

[20] P. Blackburn and J. Bos. "Representation and inference for natural language". In: *A first course in computational semantics. CSLI* (2005) (cit. on p. 114).

[21] G. Hirst. *Semantic interpretation and the resolution of ambiguity*. Cambridge University Press, 1992 (cit. on p. 91).

[22] T. Parsons. *Events in the Semantics of English*. Vol. 5. Cambridge, Ma: MIT Press, 1990 (cit. on pp. 94, 114).

[23] R. Frost and J. Launchbury. "Constructing natural language interpreters in a lazy functional language". In: *The Computer Journal* 32.2 (1989), pp. 108–121 (cit. on p. 84).

[24] M. Tomita. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. USA: Kluwer Academic Publishers, 1985 (cit. on p. 85).

[25] J. Earley. "An Efficient Context-Free Parsing Algorithm". In: *Commun. ACM* 13.2 (1970), pp. 94–102. URL: https://doi.org/10.1145/362007.362035 (cit. on p. 86).

[26] D. Davidson. "The logical form of action sentences". In: (1967) (cit. on p. 94).

Chapter 7

Accommodating Negation in an Efficient Event-Based Natural Language Query Interface to the Semantic Web

This paper was published as:

S. M. Peelar and R. A. Frost. "Accommodating Negation in an Efficient Event-based Natural Language Query Interface to the Semantic Web". In: *Proceedings of the 16th International Conference on Web Information Systems and Technologies, WEBIST 2020, Budapest, Hungary, November 3-5, 2020.* Ed. by M. Marchiori, F. J. D. Mayo, and J. Filipe. SCITEPRESS, 2020, pp. 83–92. URL: `https://doi.org/10.5220/0010148500830092`

It was featured at WEBIST 2020.

7.1 Introduction

The Semantic Web consists of a collection of triplestores accessible via *endpoints* that process queries using various query languages. Widely used methods for querying triplestores include using SPARQL [18] and Linked Data Fragments (LDF) [15].

These query languages, while powerful, are not designed with end-users in mind, with their primary use cases aimed towards databases rather than user-facing applications. An alternative approach to using a database querying language directly is to use a Natural Language Query Interface (NLQI). NLQIs have a number of benefits including being accessible through both text and speech modalities.

There are two main approaches used by NLQIs: Machine Learning (ML) can be used to attempt to determine the user's intent and retrieve corresponding relevant information. This has the advantage of being able to support a wide variety of queries, with the risk that returned information may not truly satisfy the user's intent. The second type of approach is to use a Compositional Semantics (CS) to directly answer the query with respect to a knowledge base. CS is predicated on the notion that the meaning of a sentence can be derived from the meaning of its parts [25]. This has the advantage that the answer to a query is as correct as the information in the knowledge base itself. As a result, CS-based NLQIs are able to express highly sophisticated "narrow" queries using complex linguistic constructs including superlatives and chained prepositional phrases. For example, it is possible for the NLQI presented in this paper to evaluate, with respect to a knowledge base consisting of facts about the solar system, the query:

```
which vacuumous moon that orbits the planet that is orbited by the
most moons was discovered by nicholson or pickering with a telescope
                in 1898 at not mt_wilson or not mt_hopkins
```

However, CS approaches have drawn a lot of criticism. They have been characterized as being rigid, and therefore not sufficiently able to handle complex queries in real world applications. Recent work has addressed a number of these issues, including accommodating chained complex prepositional phrases [10], n-ary transitive verbs [2] and superlative phrases [5]. It has also been shown that CS can be memoized for efficient evaluation [6], which also enables offline pre-computation of query results.

One criticism of our previous approaches was that they relied on the Open World Assumption (OWA) and hence could not support negation in queries. While the

Resource Description Framework (RDF) [11] underlying the Semantic Web itself is predicated on the OWA, there exist triplestores where the Closed World Assumption (CWA) holds, particularly in knowledge bases for expert systems. It would be ideal to support negation in queries to these triplestores.

In this paper, we show that it is possible to accommodate negation in an English NLQI to an event-based triplestore where the CWA holds. In particular, we describe an English NLQI to an event-based triplestore using a CS that supports arbitrary quantification including negation, complex linguistic constructs including chained prepositional phrases with superlatives and n-ary transitive verbs. The approach is an extension of Montague's approach [25]. Readers are directed to [5] and [6] for an introduction to the work that this paper builds on

In Section 7.2 we describe previous work on NLQIs to the Semantic Web that support negation. In Section 7.3 we describe how to access a live demonstration of our NLQI that can accommodate the queries presented in this paper along with some other example queries. In Section 7.4 we describe our event-based semantics and in Section 7.5 we describe how to accommodate negation where the CWA holds. In Section 7.6 we provide a list of examples queries and explain how they are processed. Finally, we conclude in Section 7.7 and Section 7.8.

7.2 Previous work

In 2002, Frost and Boulos introduced the notion of "complementary sets" as a way of accommodating negation in FLMS, a set-theoretic version of Montague Semantics [23]. Their approach has two drawbacks: first, that a separate denotation for transitive verbs had to be created for handling queries such as "`discover no moon`". In practice, this meant that 4 denotations of transitive verbs were required: for active and passive tense verbs, and corresponding "`no`" queries (as in "`discover no moon`" or "`discovered by no person`"). The approach presented in this paper requires only one denotation for transitive verbs for all cases, including with the presence of chained prepositional phrases and superlatives. Second, their approach required

that the cardinality of the set of entities in the database be a known constant. The denotations presented in this paper receive the cardinality of the set of entities as an argument instead. A query is made to the triplestore itself to retrieve the cardinality of the set of entities, removing the need for computing it locally.

Champollion showed that that negation in event-based CS can be accommodated using "negative events" [20]. These appear to be similar to the ideas expressed in [23], although both approaches were developed independently. Where Frost and Boulos discuss representing the result of a "negative" query by implicitly enumerating the complement of a set of entities, Champollion describes events that preclude other events from occurring. This gives some confidence about the nature of the approach taken towards accommodating negation in CS. Our own approach to negation in this paper is based in part on [23], but is event-based rather than entity-based and therefore suitable for event-based triplestores.

SQUALL [19] has limited support for negation in queries, mapping negation onto the "`NOT EXISTS`" construct of SPARQL. In particular, SQUALL has a denotation for the adverb "`not`", where its presence removes triples from the result set. This implies closed-world semantics for the query [12], although this is not discussed by the authors. SQUALL is unable to accommodate negation in noun-phrases (such as "which non-moon spins"). SQUALL is also unable to negate termphrases (for example "not Hall or not Galileo"). The reading of SQUALL queries is also not as natural as our semantics. The example given in [13], "*Which author of Paper42 has not affiliation Salford_ University?*" could be expressed in the semantics of this paper as "*Which author of Paper42 is not affiliated with Salford_ University?*". Also, where SQUALL depends strictly on translation to SPARQL, the approach described in this paper is not tied to any particular database query language or interface and could readily be adapted to relational databases.

7.3 How to Access our NLQI

A live demonstration of our NLQI is accessible via the following URL:

`https://speechweb2.cs.uwindsor.ca/solarman4/demo_sparql.html`

In addition to accepting textual input, it also can be interacted with speech on browsers that support the WebSpeech API [9]. Currently, this includes Google Chrome-based browsers and Firefox.

7.3.1 System overview

The approach presented in this paper is based on Richard Montague's denotational semantics [25]. In particular, our system derives the meaning of a query from the meaning of its parts. A query is evaluated with respect to a triplestore as though it were a formal mathematical expression using an Executable Attribute Grammar [21]. For example, the query "`ganymede discovered no moons`" is evaluated as the expression:

$$\|phobos\|\ (\|discovered\|\ (\|no\|\ \|moons\|))$$

where $\|x\|$ represents the denotation (meaning) of x.

7.3.2 Supported Features

The following are a list of example queries that demonstrate features supported by the interface.

n-ary Transitive verbs:

`who used a telescope to discover a moon`

Quantification:

`who used two telescopes to discover one moon` ⇒ *science_ team_ 2*

Chained prepositional phrases:

`which telescope was used by a person in 1877` ⇒ *refractor_ telescope_ 1*

Superlatives:

`hall discovered the most moons that orbit mars` ⇒ *True*

Negated noun-phrases:

`a non-planet was discovered` ⇒ *True*

Negated verb-phrases:

`allen did not discover anything` ⇒ *True*

Negated term-phrases, including conjunction:

`not hall and galileo discovered phobos` ⇒ *False*

Adjectives:

`enceladus is a vacuumous moon` ⇒ *True*

The above features can be combined arbitrarily to form rich queries. For example, adjectives can be combined with negation:

`mars is a non-blue planet` ⇒ *True*

In all cases, the query processor returns the syntax tree of the query to help the user understand how the query was evaluated [2]. A list of example queries and a discussion of how they are evaluated can be found in Section 7.6.

7.4 Event-Based Denotational Semantics

The approach described in this paper builds upon FLMS [24], EV-FLMS [14], UEV-FLMS [10], and most recently Memoized UEV-FLMS [6]. Notably, our semantics are event-based rather than entity-based. The fundamental data structure underlying our semantics is called the Function defined by a Relation, or *FDBR*, described in Section 7.4.2. This data structure has been shown to be useful in answering a wide variety of Natural Language queries [5]. In this paper, we show how the FDBR can be used to answer queries involving negation in event-based databases where the CWA holds.

7.4.1 Event-based Triplestores

A conventional triplestore is a database of triples that have the form *(Subject, Predicate, Object)*. An event-based triplestore is a triplestore where the *Subject* of a triple denotes an event [17][14]. The main advantage event-based triplestores offer is that it is straightforward to add additional information to an event by simply adding more triples referencing that event. It is less straightforward to do the same in a triplestore where the *Subject* denotes an entity. Such an approach in a conventional triplestore requires *reification* and involves using ontological information to link multiple triples together.

As an example, consider a triple that describes the statement "Jane bought a pencil":

<ent:Jane> <act:purchase> <ent:pencil_1> .

Without reification, there is no way to add other information about the purchase to the triplestore, such as the price, or the time or location that the transaction took place. In an event-based triplestore, this is straightforward:

<event:1> <type> <type:purchase_ev> .

<event:1> <subject> <ent:Jane> .

<event:1> <object> <ent:pencil_1> .

Since the triples directly reference the event itself, adding more information about the event simply involves adding more triples to the triplestore with the *Subject* matching the event.

7.4.2 The Function Defined by a Binary Relation (FDBR)

The notion of a *Function Defined by a Binary Relation* (FDBR) was first described in [10] as useful datastructure for accommodating chained prepositional phrases in Natural Language Queries. It was shown that the word "by", as in "discovered by", could be treated as a "virtual preposition" under this approach. In [5] it was shown that the FDBR can be used to answer many kinds of Natural Language (NL)

queries including superlatives (including those that occur in a prepositional phrase), and as a useful datastructure for memoizing the results of queries performed in the denotations. This vastly improved query execution time and opened the door for offline computation of results. The definition of the FDBR is as follows:

$$\text{FDBR}(rel) = \{(x, \text{image}_x) \mid (\exists e)\ (x,e) \in rel\ \ \&\ \ \text{image}_x = \{y \mid (x,y) \in rel\}\}$$

Where *rel* is the name of a binary relation. The FDBR has been shown to be useful for the denotation of transitive verbs. Consider the denotation for the active voice of "`discover`" given in [2], for example:

$$\begin{aligned}\|discover\| = \lambda t.\{(s,\ relevs) \mid (s,evs)\ \in\ &\text{FDBR}(\text{discover}_{rel}) \\ &\&\ (t\ \ \text{obj\_fdbr}(evs) \neq \emptyset)\ \ \&\ \ relevs = \text{gather}(\text{obj\_fdbr}(evs))\}\end{aligned}$$

where $\text{obj\_fdbr}(evs)$ is the FDBR from the objects in the events of the set *evs* to the events they participate in within *evs*. "`discover phobos`", where "`phobos`" is a proper noun, results in the FDBR:

$$\{(\text{e}_{\text{hall}}, \{\text{ev}_{1045}, \text{ev}_{1046}\})\}$$

The FDBR can be readily extended to *n*-ary relations (and hence *n*-ary transitive verbs) [2]. In this paper we show that with some small modifications, the FDBR can be used to answer NL queries with negation as well, in cases where the CWA holds.

7.5 Accommodating Negation

Negation in NL queries is only possible if the CWA holds for a database. We modify the semantics presented in [5], [7] and [2] such that the results of denotations may return the complement of an FDBR in addition to an FDBR, adopting a similar approach to [23]. We define this type as follows:

```
type Result = FDBR fdbr | ComplementFDBR fdbr
```

We then define the intersection of two `Result` types as follows:

```
intersect_result (FDBR a) (FDBR b)
  = FDBR $ intersect_fdbr a b
intersect_result (FDBR a) (ComplementFDBR b)
  = FDBR $ difference_fdbr a b
intersect_result (ComplementFDBR a) (FDBR b)
  = FDBR $ difference_fdbr b a
intersect_result (ComplementFDBR a) (ComplementFDBR b)
  = ComplementFDBR $ union_fdbr a b
```

Where `intersect_fdbr` operates as it did previously, and a new function `difference_fdbr` is introduced as follows:

$$difference\_fdbr = \lambda ms.\{(e_1, evs_2) \mid (e_1, evs_1) \in m \;\&\; (\forall (e_2, evs_2))\; ((e_2, evs_2) \in s \;\Rightarrow e_1 \neq e_2)\}$$

That is, *difference_fdbr* removes all entities found in the left column of the second FDBR from the first FDBR. This is a key function for performing negation, and plays a similar role to the "`NOT EXISTS`" operator in SPARQL. A function is introduced for computing the union of `Results` as well:

```
union_result (FDBR a) (FDBR b)
  = FDBR $ union_fdbr a b
union_result (FDBR a) (ComplementFDBR b)
  = ComplementFDBR $ b `difference_fdbr` a
union_result (ComplementFDBR a) (FDBR b)
  = ComplementFDBR $ a `difference_fdbr` b
union_result (ComplementFDBR a) (ComplementFDBR b)
  = ComplementFDBR $ a `intersect_fdbr` b
```

This is used in the denotation of "`and`" and "`or`" as used with termphrases. Next, we introduce a function to obtain the cardinality of a Result:

```
cardinality _ (FDBR np) = List.length np
cardinality (Just num_ents) (ComplementFDBR np)
= num_ents - length np
```

The first argument to this function is passed in from the query pipeline described in [6], and is either `Nothing` or `Just num_ents`, where `num_ents` is the cardinality of the set of entities in the triplestore. It will only be retrieved if the query has any denotations involving negation in it.

Our approach maintains leftmost-outermost scoping of quantifiers including negation, which enables a natural reading of the query.

7.5.1 Quantifiers

We modify the denotations of all quantifiers to be characterized in terms of the cardinality:

```
a' = intersect_result
every'' cardinality nph vbph =
  if cardinality result == cardinality nph
  then result else FDBR []
one'' cardinality nph vbph =
  if cardinality result == 1 then result else FDBR []
two'' cardinality nph vbph =
  if cardinality result == 2 then result else FDBR []
most'' cardinality nph vbph =
  if n_nph /= 0 && (n_nph_v / n_nph) > 0.5
    then result else FDBR []
  where
    n_nph = fromIntegral $ cardinality nph
    n_nph_v = fromIntegral $ cardinality res
```

where in the above denotations, `result = intersect_result'' nph vbph`. Curiously, the function `cardinality` appears as the first argument to these quantifiers, giving them three arguments in total. This function is passed in from the caller as a function that can be used to obtain the cardinality of a FDBR. A function, "`apply_card`" is used to automatically apply the cardinality function to the denotations. For example:

```
every' = applyCard every'' >|< GettsIntersect GI_Every
every = wrapS2 every'
```

The `>|<` operator is described in more detail in [6]. It is used to assign a unique name to the denotations according to the syntax tree of the query. This is useful for memoization and query optimization. "`no`" is denoted as follows:

```
no'' cardinality nph (FDBR []) = ComplementFDBR []
no'' cardinality nph vbph = if cardinality result == 0 then vbph
    else FDBR []
```

This is a departure from the denotation of "`no`" given in [23]. Namely, the complement of the empty FDBR (denoting "everything") is returned when an empty FDBR is passed as the second argument to "`no`". This is critical in handling "`no`" in the denotation of transitive verbs as discussed later in Section 7.5.4.

7.5.2 Negating Noun- and Verb-phrases

We denote "`not`", when applied to a verb-phrase (such as "`not spins`") as follows:

```
not (FDBR vbph) = ComplementFDBR vbph
not (ComplementFDBR vbph) = FDBR vbph
```

"`non`" plays a similar role as a prefix to a noun-phrase, and can be denoted as:

```
non = not
```

7.5.3 Negating Term-phrases

One aspect missing from both [13] and [23] is the notion of a negated termphrase, for example "`not hall`", "`not a moon`", "`not one moon`" and "`not no moon`", which exhibits double negation. Negating termphrases provides more flexibility to the query interface, making it possible to express the query:

```
who discovered in 1877 not one moon that orbits mars
```

Where "`one`" denotes "exactly one". This query explicitly is excluding any discoverers that discovered exactly one *moon* that orbits *mars*. It results in *hall*, because *hall* discovered two *moons* that orbit *mars* in 1877. "`not`" when applied to a term-phrase, such as "`hall`" or "`a moon`" is denoted as follows:

```
termnot tmph vbph = intersect_result (not (tmph vbph)) vbph
```

Therefore `not hall spins` is evaluated as follows:

```
    (not hall) spins
⟹ intersect_result (not (hall spins)) spins
⟹ intersect_result (not (FDBR [])) spins
⟹ intersect_result (ComplementFDBR []) spins
⟹ spins
⟹ True  (because spins is not empty)
```

Negating term-phrases was not discussed in [23], and it offers more flexibility in the nature of queries that can be performed (see Section 7.6)

7.5.4 A Denotation for Transitive Verbs that Accommodates Superlatives, Prepositional Phrases, and Negation

Transitive verbs are less straightforward to accommodate with negation. Consider the following query:

```
ganymede discovered no moons
```

This query should evaluate to *True*, as *ganymede*, a *moon*, was not the subject of any discovery events – however, *ganymede* is not the ***subject*** of any events of type "***discover***". Therefore, it is missing from $\mathsf{FDBR}(\mathsf{discover}_{rel:subject})$, where $\mathsf{discover}_{rel:subject}$ is the relation from the *subject*s of the ***discover*** events to the events of type "***discover***" that they participate in.

A denotation is given in [23] that accommodates this usage of transitive verbs; however it requires syntactic disambiguation at the grammar level to apply correctly. The approach also does not scale well when other linguistic constructs are introduced, such as chained prepositional phrases and superlatives, requiring a new denotation to support each usage. The examples given in [23] required 4 denotations for transitive verb depending on the context.

The denotation we introduce expands on the denotation introduced in [5], where we described how superlative phrases can also be accommodated. This new denotation evaluates the list of prepositional phrases (including superlatives) in leftmost-outermost order, which is consistent with other work in the area [22], [13]. For example, the query "`discovered a moon in 1877 with a telescope`" would be evaluated with scoping as though it were as follows: "`discovered (a moon (in 1877 (with a telescope)))`" – that is, "`with a telescope`" takes precedence over "`in 1877`", which in turn takes precedence over "`a moon`".

Only one denotation for transitive verbs is required for all cases (rather than 4 as in [23]). In particular, the word "`no`" can be handled compositionally rather than syntactically in the query.

We modify the denotation for transitive verbs given in [6] to evaluate the list of prepositional phrases in leftmost-outermost order. The `filter_ev` function, described in [10], is modified to operate on one prepositional phrase at a time: a new FDBR is computed for each prepositional phrase applied. This allows superlatives to be neatly evaluated in the order they appear rather than in a separate stage after the prepositions are evaluated as denoted in [5]. `filter_ev` also is modified to account for negation in the query: first, the current prepositional phrase is evaluated against the empty FDBR (`FDBR []`). If the result is not an FDBR, then it is a "`no`"

termphrase:

`in no place (FDBR []) ⟹ ComplementFDBR []`

This is owing to the denotation of "`no`" used in Section 7.5.1. Indeed, the only way to obtain a non-empty FDBR from applying an empty FDBR is through negation. When this is the case, `filter_ev` returns a complement in the same fashion as [23].

Since `filter_ev` can also receive the complement of an FDBR, as in the case when negation is present in the query, applying term-phrases can be difficult. The complement operation is reversed by taking the FDBR of the transitive verb itself and performing the intersection of it with the complement passed into `filter_ev`. Whether negation is present in the current prepositional phrase or not, this FDBR is passed in to the termphrase of that preposition. If no negation is present in the current preposition, the result is returned as-is, unless it contains a superlative. Otherwise, if negation is present, then if a complement of an FDBR was passed into `filter_ev`, the FDBR used in the denotation of the transitive verb itself is used to compute the complement, otherwise the FDBR passed into `filter_ev` is used directly. This allows for "`discover no moon in 1948`" to work as expected. This can neatly handle the following cases:

```
discover no moons in no places with no telescopes
```

The result is the complement of the FDBR of those that discovered a moon in a place with a telescope

```
discover no moons in 1877
```

The result is the complement of the FDBR of those that discovered a moon in 1877

```
discover a moon in 1877
```

The result is the FDBR of the people that discovered a moon in 1877.

7.5.5 Obtaining the cardinality of the entities of the triplestore

In systems where the Open World Assumption holds, obtaining the cardinality of the set of entities may not be possible, as the cardinality of that set may be infinite. Attempting to obtain that set at all may not be practical. Even in systems where the CWA holds, obtaining the set of all entities in the database may not be feasible. Fortunately, only the cardinality is required to start answering queries.

A new querying primitive is introduced from [6] that queries the remote triplestore itself for the cardinality of the set of entities in the triplestore:

```
getts_cardinality_allents ev_data props
```

Here, `ev_data` represents the URL of the triplestore itself (in the case of SPARQL, a SPARQL endpoint URL), and `props` is the set of properties of the events contained in the triplestore whose entities should be counted towards the cardinality. In the example queries given in this paper, the properties listed for cardinality are "*subject*", "*object*", "*location*", and "*implement*". We exclude the "*year*" property as all entities must exist both physically and temporally [11]. This function, like the other `getts*` family functions described in [5], can be specialized for different types of databases, including relational triplestores.

This alleviates having to send the full set of entities to the semantics in order to answer a query that uses negation. Note that the cardinality of the set of entities of the triplestore is only ever required in queries that have negation present. Using the memoization and triplestore querying framework described in [6], a guarantee is made that if no negation is present in the query, the cardinality query will never be performed. Therefore, our denotations for negation in queries, including "`not`", "`non`", "`no`" and "`the least`", are drop-in enhancements to NLQIs built using our framework: if the CWA holds for the application, all one needs to do is add these denotations in. Otherwise, the NLQI will operate with the open world semantics described in [5].

In some cases, the user may want to force evaluation of the complement, for

example when they know the result set will be small. It is possible to introduce a special denotation, "`force_eval`", will obtain all triples and force retrieval of all entities. This may be cached on the interface to alleviate the load against the remote triplestore using the memoization framework in [5]. It may be appropriate to evaluate the complement if its cardinality is under a certain threshold as well, triggering "`force_eval`" automatically – this could be customized on a per-application basis.

7.5.6 Accommodating "the least"

In [5] we described how to accommodate superlative phrases compositionally by delegating their evaluation to the transitive verb they are arguments of. This allows them to appear in chained prepositional phrases.

One problem described with that approach was answering queries with superlatives such as "the least" or "the lowest number of". The main reason for this was owing to the OWA underlying the semantics. Under that approach, "`which planets are orbited by the least number of moons`" would return *earth*, despite both *venus* and *mercury* having a lower number of moons than Earth. The semantics had no concept of *zero* and could only report about what was observable. Since there were no events explicitly stating that *venus* and *mercury* had no moons, it could not assume that it was not the case.

We propose an alternative approach in this paper, where "`the least`" is handled similarly to the word "`no`". The denotation for "`the least`" first checks that the complement of the FDBR is non-empty. If so, "`the least`" returns the complement of that FDBR – this allows for "venus" and "mercury" to appear in the result set while removing all non-candidates. If the complement of the FDBR is empty, however, then it performs the same cardinality partitioning that the "`the most`" does [5], except it chooses the lowest *object cardinality* entities to form the result rather than the greatest.

7.6 Example Queries

The following are some example queries that can be handled by our NLQI. With each query we explain the result and how it was evaluated.

`no people spin` ⇒ *True*

The intersection of people_{FDBR} and spins_{FDBR} is empty, therefore `no` returns `spin`, which is non-empty and therefore *True*.

`a non person exists` ⇒ *True*

"`non person`" is the complement of person_{FDBR}, and the intersection of this complement with exists_{FDBR} (which is the complement of the empty FDBR) is the same as the complement of the union of person_{FDBR} with the empty FDBR. The answer is characterized in terms of the cardinality, which for the complement of an FDBR is defined as the cardinality of the number of set of entities in the triplestore minus the cardinality of the FDBR itself. This is greater than 0, and therefore there is at least one entity that is both a non-person and exists.

`a person does not exist` ⇒ *False*

This computes the intersection of person_{FDBR} with the complement of exists_{FDBR}, which is just the empty FDBR. Therefore, the result is empty, and the answer is *False*.

`what discovered no moon in 1877` ⇒ *everything except: hall*

This sentence is treated similarly to "`what did not discover a moon in 1877`". The result is the complement of the set of entities that discovered a moon in 1877, in this case, *hall*.

`what discovered a non moon` ⇒ *nothing.*

This query is specifically asking about entities that discovered non-moons – the entities that did not discover anything are not included in this set and therefore an FDBR is returned. Since that FDBR is empty, the result is that nothing in our triplestore discovered any non-moons.

`allen discovered no moon at no places` ⇒ *True*

The result of "`discovered no moon at no places`" is the complement of the FDBR returned from "`discovered a moon at a place`". This includes entities that either discovered a moon at no known location, or discovered a non-moon at a known location. Since *allen* does not appear in the FDBR returned by "`discovered a moon at a place`", the result is *True*.

`what discovered the most moons using no telescopes`
⇒ *voyager_science_team*

This query combines both a superlative phrase with negation. The query is asking "out of the events where entities discovered something without using a telescope, which ones discovered the most moons". Since `voyager_science_team` used no telescopes at all to discover 22 moons, more than any other entities that discovered using no telescopes, they are in the result set.

`what was discovered by no team in 1877` ⇒ *everything.*

This query is handled the same as "`what was not discovered by a team in 1877`", which returns the complement of the empty FDBR, since no *teams* discovered anything in 1877.

`how was something discovered using no telescope` ⇒ *I can't perform this query because I would need to enumerate the entire triplestore.*

This query is asking about which implements that are not telescopes were used in a discovery event. However, "`something`" is defined as the complement of the empty FDBR, and "`discovered using no telescope`" is the complement of the FDBR of "`discovered using a telescope`". Since the intersection of the two complements is itself a complement, "`how`" receives the complement of an FDBR and is unable to enumerate the events to retrieve implements from directly. Although it is possible to answer the query by fully evaluating the complement, we have not implemented this behaviour in our NLQI at this time. However, a similar query, "`which non telescope was used to discover something`" is able to yield the result "*cassini,*

voyager_1, voyager_2".

`not hall discovered ganymede` ⇒ *True*

"`not hall`" is a negated term-phrase. The result is *True* because *galileo* discovered *ganymede*, not *hall*.

`which person that does not spin discovered no planet in 1877 using a telescope and is a discoverer` ⇒ *bernard, bond, cassini, christy, dollfus, galileo, hall, herschel, holman, huygens, karkoschka, kowal, kuiper, lassell, melotte, nicholson, perrine, pickering, sheppard, showalter*

The result is all of the people that are *discoverers*, since none of the members of *person spin*, and none of them discovered a *planet* in 1877 using a *telescope*. It may be helpful to examine the scoping of this query:

```
which (person `that` (does not spin)) ((discovered (no planet) [in
        1877, using (a telescope)]) and (is a discoverer))
```

`nothing exists` ⇒ *False*

This is *False* because the intersection of *thing* and *exists* is non-empty.

`everything exists` ⇒ *True*

This is *True* because *thing* and *exists* are both the complement of the empty FDBR, and the intersection of those results in the same. Therefore *thing* is a subset of *exists*.

`what was not discovered by hall` ⇒ *everything except: deimos, phobos*

This is the complement of the FDBR "`discovered by hall`". The answer is the set of all things excluding those that *hall* discovered.

`phobos and deimos were not discovered by not hall` ⇒ *True*

This query features double negation and is equivalent to asking "`phobos and deimos were discovered by hall`". The result is *True*.

`not not kuiper discovered not not nereid` ⇒ *True*

This query also features double negation on the termphrases *kuiper* and *nereid*. This is equivalent to the query `kuiper discovered nereid`.

`which non vacuumous moon that orbits most planets that spin was not discovered by kuiper at two places using the most telescopes in 1942` ⇒ *none.*

This query features a variety of complex linguistic constructs, including nested *n*-ary transitive verbs, adjectives, negation, chained prepositional phrases, quantification and superlative phrases. The result is "*none*" because "`non vacuumous moon that orbits most planets that spin`" returns the empty FDBR, and the intersection of the empty FDBR with any FDBR is also the empty FDBR.

`not no moon orbits mars` ⇒ *True*

This query features a negated termphrase, which itself consists of the word "`no`" (entailing negation itself). "`orbits mars`" is the FDBR from the entities *phobos* and *deimos* to their orbit events, and "`not no moon orbits mars`" evaluates to "`orbits mars`" with the entities of "`no moon orbits mars`" removed. Since "`no moon orbits mars`" is *False*, it returns the empty FDBR, which is then removed from "`orbits mars`", giving a non-empty result. Therefore, the query returns *True*. This provides evidence that our NLQI correctly handles negation as a compositional construct.

`who discovered no moons at no places` ⇒ *allen, baum, buie, burns ...*(full results omitted here)*... weaver, young_ e_ f, young_ l_ a*

The result is everyone that is not known to have discovered a moon at a place. This includes the discoverers that discovered a moon at no known location, or whose location property is not listed in the event, or discoverers that discovered a non moon at a known location.

7.7 Future Work

Our next efforts will be focused on creating an NLQI to DBPedia using the approaches described here and in [6]. Specifically, we plan to use Timbr.ai [4] to provide a relational view of DBPedia, targeting SQL as the query language. Once this is done, we plan to test our NLQI using well-known benchmarks such as QALD [8].

We also plan to explore interfacing with non-event based triplestores in general. ML approaches may be useful in contexts where ontological information is not available for reification.

7.8 Conclusions

We have shown that it is possible to accommodate negation in our event-based CS efficiently. We have shown that our approach to negation is powerful, able to be applied to noun-phrases, verb-phrases, and term-phrases. We presented a denotation for "`no`" that enables it to be treated as a quantifier that can be compositionally used in conjunction with transitive verbs, either as an argument to the verb or as a preposition. We improved on [23] by maintaining only one denotation for transitive verbs throughout the semantics rather than requiring different denotations depending on the context. Notably, our approach to negation seems to be consistent with other work in event semantics [20]. We improved on [16] by enabling the negation of term-phrases, and also enabling our approach to be used with other query languages than SPARQL. We discussed the necessity of the Closed World Assumption for queries involving negation and described how to extend the CS given by Frost and Peelar in [5] to accommodate negation in queries.

Where the CWA is not appropriate, leaving out the denotations for "`not`", "`non`", and "`the least`" is sufficient to restore the Open World Assumption in the semantics. Our approach also fits within the memoization framework in [5]. We also discussed example queries that are supported with our NLQI and explained how

the results are formed. We believe now that our semantics is ready to be benchmarked directly against other systems on large knowledge bases using, for example, QALD-9 [8].

Acknowledgements

This research was supported by NSERC of Canada.

Supplementary Material

The complete source code for the demonstration, including the semantics and parsing framework, can be found online at the Hackage Haskell package repository under the *XSaiga* project [1]:

`https://hackage.haskell.org/package/XSaiga`

Bibliography

[1] R. Hafiz, R. Frost, S. Peelar, P. Callaghan, and E. Matthews. *The XSAIGA Package*. https://hackage.haskell.org/package/XSaiga, 2020 (cit. on p. 142).

[2] S. M. Peelar and R. A. Frost. "A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore". In: *IEEE 14th International Conference on Semantic Computing, ICSC 2020, San Diego, CA, USA, February 3-5, 2020*. IEEE, 2020, pp. 257–262. URL: `https://doi.org/10.1109/ICSC.2020.00054` (cit. on pp. 122, 126, 128).

[3] S. M. Peelar and R. A. Frost. "Accommodating Negation in an Efficient Event-based Natural Language Query Interface to the Semantic Web". In: *Proceedings of the 16th International Conference on Web Information Systems and Technologies, WEBIST 2020, Budapest, Hungary, November 3-5, 2020*. Ed. by M. Marchiori, F. J. D. Mayo, and J. Filipe. SCITEPRESS, 2020, pp. 83–92. URL: `https://doi.org/10.5220/0010148500830092` (cit. on p. 121).

[4] Timbr. "Timbr.ai". In: *https://wiki.dbpedia.org/timbr* (2020) (cit. on p. 141).

[5] R. A. Frost and S. M. Peelar. "A New Data Structure for Processing Natural Language Database Queries". In: *Proceedings of the 15th International Conference on Web Information Systems and Technologies, WEBIST 2019, Vienna, Austria, September 18-20, 2019*. 2019, pp. 80–87. URL: https://doi.org/10.5220/0008124300800087 (cit. on pp. 122, 123, 126–128, 133, 135, 136, 141).

[6] S. M. Peelar and R. A. Frost. "A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores". In: *Web Information Systems and Technologies - 15th International Conference, WEBIST 2019, Vienna, Austria, September 18-20, 2019, Revised Selected Papers*. Ed. by A. Bozzon, F. J. D. Mayo, and J. Filipe. Vol. 399. Lecture Notes in Business Information Processing. Springer, 2019, pp. 168–194. URL: https://doi.org/10.1007/978-3-030-61750-9%5C_8 (cit. on pp. 122, 123, 126, 130, 131, 133, 135, 141).

[7] R. A. Frost and S. M. Peelar. "An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore". In: *Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, October 8th - 9th, 2018*. Ed. by K.-S. Choi, L. E. Anke, T. Declerck, D. Gromann, J.-D. Kim, A.-C. N. Ngomo, M. Saleem, and R. Usbeck. Vol. 2241. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 1–16. URL: http://ceur-ws.org/Vol-2241/paper-01.pdf (cit. on p. 128).

[8] R. Usbeck, R. H. Gusmita, A.-C. N. Ngomo, and M. Saleem. "9th challenge on question answering over linked data (QALD-9)". In: *Semdeep/NLIWoD@ ISWC*. 2018, pp. 58–64 (cit. on pp. 141, 142).

[9] W3C et al. "Web Speech API". In: *Retrieved from World Wide Web Consortium: https://w3c. github. io/speech-api* (2018) (cit. on p. 125).

[10] S. Peelar. "Accommodating prepositional phrases in a highly modular natural language query interface to semantic web triplestores using a novel event-based denotational semantics for English and a set of functional parser combinators". MA thesis. University of Windsor (Canada), 2016 (cit. on pp. 122, 126, 127, 133).

[11] T. W. W. W. C. (W3C). *RDF 1.1 Semantics*. `https://www.w3.org/TR/rdf11-mt/`. [Online; accessed 06-September-2016]. 2014 (cit. on pp. 123, 135).

[12] F. Darari, S. Razniewski, and W. Nutt. "Bridging the Semantic Gap between RDF and SPARQL using Completeness Statements [Extended Version]". In: *CoRR* abs/1408.6395 (2014). arXiv: `1408.6395`. URL: `http://arxiv.org/abs/1408.6395` (cit. on p. 124).

[13] S. Ferré. "SQUALL: The expressiveness of SPARQL 1.1 made available as a controlled natural language". In: *Data & Knowledge Engineering* 94 (2014), pp. 163–188 (cit. on pp. 124, 132, 133).

[14] R. A. Frost, J. Donais, E. Matthews, and W. Agboola. "A Demonstration of a Natural Language Query Interface to an Event-Based Semantic Web Triplestore". In: *ESWC*. Springer LNCS Volume 8798. 2014, pp. 343–348 (cit. on pp. 126, 127).

[15] R. Verborgh, M. Vander Sande, P. Colpaert, S. Coppens, E. Mannens, and R. Van de Walle. "Web-Scale Querying through Linked Data Fragments." In: *LDOW*. Citeseer. 2014 (cit. on p. 121).

[16] S. Ferré. "SQUALL: a controlled natural language as expressive as SPARQL 1.1". In: *International Conference on Application of Natural Language to Information Systems*. Springer. 2013, pp. 114–125 (cit. on p. 141).

[17] R. A. Frost, B. S. Amour, and R. Fortier. "An Event Based Denotational Semantics for Natural Language Queries to Data Represented in Triple Stores". In: *ICSC, 2013 IEEE Seventh International Conference on Semantic Computing*. IEEE. 2013, pp. 142–145 (cit. on p. 127).

[18] S. Harris, A. Seaborne, and E. Prud'hommeaux. "SPARQL 1.1 query language". In: *W3C recommendation* 21.10 (2013) (cit. on p. 121).

[19] S. Ferre. "SQUALL: A Controlled Natural Language for Querying and Updating RDF Graphs". In: *proc. of CNL 2012*. LNCS 7427. 2012, pp. 11–25 (cit. on p. 124).

[20] L. Champollion. "Quantification and negation in event semantics". In: (2011) (cit. on pp. 124, 141).

[21] R. Hafiz. "Executable Attribute Grammars for Modular and Efficient Natural Language Processing". In: (2011) (cit. on p. 125).

[22] L. Champollion. "Quantification in Event Semantics". In: *Talk given at the 6th int. symp. of cogn., Logic and commun.* 2010 (cit. on p. 133).

[23] R. A. Frost and P. Boulos. "An Efficient Compositional Semantics for Natural-Language Database Queries with Arbitrarily-Nested Quantification and Negation". In: *Advances in Artificial Intelligence, 15th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2002, Calgary, Canada, May 27-29, 2002, Proceedings*. Ed. by R. Cohen and B. Spencer. Vol. 2338. Lecture Notes in Computer Science. Springer, 2002, pp. 252–267. URL: https://doi.org/10.1007/3-540-47922-8%5C_21 (cit. on pp. 123, 124, 128, 131–134, 141).

[24] R. Frost and J. Launchbury. "Constructing natural language interpreters in a lazy functional language". In: *The Computer Journal* 32.2 (1989), pp. 108–121 (cit. on p. 126).

[25] D. Dowty, R. Wall, and S. Peters. *Introduction to Montague Semantics*. Dordrecht, Boston, Lancaster, Tokyo: D. Reidel Publishing Company, 1981 (cit. on pp. 122, 123, 125).

Chapter 8

Conclusions

8.1 Future Work

In the future we aim to support non event-based triplestores in addition to event-based triplestores and relational databases. In particular, we are interested in building a NLQI to DBPedia to directly evaluate our approach against other NLQIs to the Semantic Web. We intend to conduct a formal user study to meet this goal, including using established benchmarks such as QALD-9 [9] to conduct quantitative comparisons.

8.2 Conclusions

We have shown that a scalable, efficient, expressive and precise method for processing natural-language queries to the Semantic Web can be built using a Compositional Semantics (CS). We have shown many features of English that are non-compositional can in fact be handled compositionally within a NLQI, addressing the Expressiveness and Precision aspects of the thesis statement. This is owing to the use of a datastructure called the Function Defined by a Binary Relation (FDBR). We have shown how it can be used to answer queries with traditionally "non-compositional" features in a CS such as those including superlatives, comparatives, n-ary transitive verbs and chained prepositional phrases. Our approach is

highly tolerant of both syntactic and semantic ambiguity. We have also addressed how to accommodate negation in queries to triplestores where the Closed World Assumption holds. As these features of our query processor are implemented compositionally, they can be combined in queries arbitrarily.

We have described a framework for evaluating CS efficiently through the use of memoization, drastically improving query evaluation computational complexity. This same framework provides a means to efficiently form a minimal set of queries of information needed from a triplestore to answer a query, critically keeping the event semantics distinct from the triplestore querying process itself. This allows the event semantics to be used with a wide variety of database query languages and paradigms such as SPARQL, Triple Pattern Fragments, and even SQL with relational databases. This satisfies the Scalability and Efficiency aspects of the thesis statement.

We have shown our approach can be used in highly power constrained environments. One area where our approach could be useful is in constructing NLQIs to the Internet of Things. This could substantially benefit users with certain disabilities, providing modalities such as speech and text to common household items that otherwise may not be very accessible. We have also shown that our approach is able to be used directly in the web browser, where there are no intermediate servers required to process a Semantic Web NL query. This could be seen as a first step towards treating the Semantic Web as an accessible extension of the World Wide Web.

Appendices

Appendix A - Source code

The source code for Solarman and the XSaiga parser can be obtained online via this URL:

`https://hackage.haskell.org/package/XSaiga-1.7.0.0/XSaiga-1.7.0.0.tar.gz`

The XSaiga package for Haskell is available online at this URL:

`https://hackage.haskell.org/package/XSaiga`

Appendix B - List of Refereed Papers Relating to the Thesis

Below is a list of peer-reviewed conference papers that I have authored or co-authored, which are related to this Thesis. Electronic versions of this papers can be found online:

[12] J. A. Donais, R. A. Frost, S. M. Peelar, and R. A. Roddy. "A system for the automated author attribution of text and instant messages". In: *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM. 2013, pp. 1484–1485

[10] S. Peelar and P. Preney. "The Windsor Build and Testing Framework". In: *Proceedings of the 5th International Workshop on OpenCL - IWOCL 2017*. ACM Press, 2017. URL: `https://doi.org/10.1145%2F3078155.3078184`

[8] S. Peelar, J. Urbanic, R. Hedrick, and L. Rueda. "Real-Time Visualization

of Bead Based Additive Manufacturing Toolpaths Using Implicit Boundary Representations". In: *Proceedings of CAD'18*. CAD, 2018-07. URL: https://doi.org/10.14733%2Fcadconfp.2018.327-331

[7] R. A. Frost and S. M. Peelar. "An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore". In: *Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, October 8th - 9th, 2018*. Ed. by K.-S. Choi, L. E. Anke, T. Declerck, D. Gromann, J.-D. Kim, A.-C. N. Ngomo, M. Saleem, and R. Usbeck. Vol. 2241. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 1–16. URL: http://ceur-ws.org/Vol-2241/paper-01.pdf

[4] R. A. Frost and S. M. Peelar. "A New Data Structure for Processing Natural Language Database Queries". In: *Proceedings of the 15th International Conference on Web Information Systems and Technologies, WEBIST 2019, Vienna, Austria, September 18-20, 2019*. 2019, pp. 80–87. URL: https://doi.org/10.5220/0008124300800087

[2] S. M. Peelar and R. A. Frost. "A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore". In: *IEEE 14th International Conference on Semantic Computing, ICSC 2020, San Diego, CA, USA, February 3-5, 2020*. IEEE, 2020, pp. 257–262. URL: https://doi.org/10.1109/ICSC.2020.00054

[3] S. M. Peelar and R. A. Frost. "Accommodating Negation in an Efficient Event-based Natural Language Query Interface to the Semantic Web". In: *Proceedings of the 16th International Conference on Web Information Systems and Technologies, WEBIST 2020, Budapest, Hungary, November 3-5, 2020*. Ed. by M. Marchiori, F. J. D. Mayo, and J. Filipe. SCITEPRESS, 2020, pp. 83–92. URL: https://doi.org/10.5220/0010148500830092

[11] R. A. Frost, J. Donais, E. Matthews, and W. Agboola. "A Demonstration of

a Natural Language Query Interface to an Event-Based Semantic Web Triplestore". In: *ESWC*. Springer LNCS Volume 8798. 2014, pp. 343–348

[11] (Frost, Donais, Matthews, Agboola, and Stewart) describes the demonstration of the Haskell program which I wrote and which forms the basis of this thesis work. The reason that I am not listed as an author is that the paper was submitted after I officially joined the research team and I was an undergraduate student at the time. I developed the Haskell program after the paper was submitted. The online program was the one used by Dr. Frost in the demonstration he gave at the conference this paper was presented at.

My contributions to the research project included:

- Improving the efficiency of the programs which implement the event-based semantics
- Integrating the event-based semantics with the parser combinators to build the query processor
- Enhancing the existing module to access the external triplestore with efficient methods to do so, including a basic form of query fusion in the form of memoization
- Demonstrating a novel method of handling the word "by" in prepositional phrases, and extending prepositional phrases to span multiple property names
- Building a web interface to the query processor which includes both an English Natural Language Interface and also a safe Direct Query Interface for directly evaluating the combinators
- Converting the parser Hafiz wrote[13] to natively work with monads in Haskell, as well as the original semantics[11] to be monad based
- Maintaining the XSaiga package on *Hackage*[1], an online repository of Haskell libraries and programs, which contains the semantics, parser, and triplestore described in this Thesis

Peer reviewed Journal papers and Book Chapters:

[5] S. Peelar, J. Urbanic, R. Hedrick, and L. Rueda. "Real-Time Visualization Of Bead Based Additive Manufacturing Toolpaths using Implicit Boundary Representations". In: *Computer-Aided Design and Applications* 16.5 (2019-01), pp. 904–922. URL: https://doi.org/10.14733%2Fcadaps.2019.904-922

[6] S. M. Peelar and R. A. Frost. "A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores". In: *Web Information Systems and Technologies - 15th International Conference, WEBIST 2019, Vienna, Austria, September 18-20, 2019, Revised Selected Papers*. Ed. by A. Bozzon, F. J. D. Mayo, and J. Filipe. Vol. 399. Lecture Notes in Business Information Processing. Springer, 2019, pp. 168–194. URL: https://doi.org/10.1007/978-3-030-61750-9%5C_8

[12] describes results of my Undergraduate thesis that I worked on with my colleagues under Dr. Richard Frost. Although I had some experience with NL applications, I had none with Machine Learning (ML) approaches at the time. Through this project I learned about Bayesian classifiers and I was able to broaden my understanding of the NLP domain.

[5, 8, 10] describes work I undertook as part of an OCE TalentEdge Internship grant for developing a simulator for Additive Manufacturing (AM) parts. I undertook this work in part to learn about High Performance Computing applications seeking to better scale the semantics presented in this dissertation for extremely large triplestores on the Semantic Web. Although the work focuses on Engineering applications, many of the heterogeneous programming principles apply.

Appendix C - Copyright Releases

1. Chapter 3 includes a refereed paper, which was published as:

   R. A. Frost and S. M. Peelar. "An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore". In: *Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Ques-*

tion Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, October 8th - 9th, 2018. Ed. by K.-S. Choi, L. E. Anke, T. Declerck, D. Gromann, J.-D. Kim, A.-C. N. Ngomo, M. Saleem, and R. Usbeck. Vol. 2241. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 1–16. URL: http://ceur-ws.org/Vol-2241/paper-01.pdf

Below is the copyright information from IEEE:

2018 IEEE. Reprinted, with permission, from Richard A. Frost and Shane Peelar, An Extensible Natural-Language Query Interface to an Event-Based Semantic Web Triplestore, October 2018.

2. Chapter 4 includes a refereed paper, which was published as:

   R. A. Frost and S. M. Peelar. "A New Data Structure for Processing Natural Language Database Queries". In: *Proceedings of the 15th International Conference on Web Information Systems and Technologies, WEBIST 2019, Vienna, Austria, September 18-20, 2019*. 2019, pp. 80–87. URL: https://doi.org/10.5220/0008124300800087

   Dr. Vitor Pedrosa, the chair of INSTICC, which maintains the publication for the paper, has personally sent the following e-mail to the authors regarding the copyright:

   Dear Shane Peelar,

   Thank you very much for your email.

   I hereby grant you the necessary authorization to add the mention paper to your doctoral dissertation as long as all the bibliography information from its publication is there too and the version that you use is the one published.

   Best regards,

   Vitor Pedrosa

3. Chapter 5 includes a refereed paper, which was published as:

   S. M. Peelar and R. A. Frost. "A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore". In: *IEEE 14th International Conference on Semantic Computing, ICSC 2020, San Diego, CA, USA, February 3-5, 2020*. IEEE, 2020, pp. 257–262. URL: https://doi.org/10.1109/ICSC.2020.00054

   Below is the copyright information from IEEE:

   2020 IEEE. Reprinted, with permission, from Shane Peelar and Richard A. Frost, A Compositional Semantics for a Wide-Coverage Natural-Language Query Interface to a Semantic Web Triplestore, February 2020.

4. Chapter 6 includes an extended revised paper that has been published as a chapter in the WEBIST 2019 Springer Book:

   S. M. Peelar and R. A. Frost. "A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores". In: *Web Information Systems and Technologies - 15th International Conference, WEBIST 2019, Vienna, Austria, September 18-20, 2019, Revised Selected Papers*. Ed. by A. Bozzon, F. J. D. Mayo, and J. Filipe. Vol. 399. Lecture Notes in Business Information Processing. Springer, 2019, pp. 168–194. URL: https://doi.org/10.1007/978-3-030-61750-9%5C_8

   Below is the copyright information from Springer:

   Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer WEBIST 2019 A New Approach for Processing Natural-Language Queries to Semantic Web Triplestores, Shane Peelar and Richard A. Frost, Copyright 2020

5. Chapter 7 includes a refereed paper, which was published as:

   S. M. Peelar and R. A. Frost. "Accommodating Negation in an Efficient Event-based Natural Language Query Interface to the Semantic Web". In: *Proceedings of the 16th International Conference on Web Information Systems*

and Technologies, WEBIST 2020, Budapest, Hungary, November 3-5, 2020. Ed. by M. Marchiori, F. J. D. Mayo, and J. Filipe. SCITEPRESS, 2020, pp. 83–92. URL: https://doi.org/10.5220/0010148500830092

Dr. Vitor Pedrosa, the chair of INSTICC, which maintains the publication for the paper, has personally sent the following e-mail to the authors regarding the copyright:

Dear Shane Peelar,

Thank you for your email.

You do have permission to do that without a problem as long as the full credit is mentioned, as you already say in your email.

Best regards,

Vitor Pedrosa

Vita Auctoris

Shane Peelar was born in 1990 in Windsor, Ontario. He completed his undergraduate degree in Computer Science from the University of Windsor in 2014, graduating with Honours and specializing in Software Engineering. He went on to complete his Masters degree in Computer Science from the University of Windsor in 2016, and in 2020 completed his Doctor of Philosophy in Computer Science from the University of Windsor.