Neurosymbolic Learning and Reasoning with OWL-based Knowledge Graphs

David Herron

Supervisors:
Dr. Ernesto Jiménez-Ruiz, Dr. Giacomo Tarroni, and Dr. Tillman Weyde

A thesis
submitted in partial fulfillment of the requirements
of the Degree of Doctor of Philosophy

School of Science and Technology
Department of Computer Science
City St George's, University of London

January, 2025

Acknowledgements

I thank my supervisors for providing a magic mix of guidance, freedom, inspiration, and motivation. My research would not be here without them, and the journey not nearly so enjoyable.

I thank Dr. Ranjay Krishna, one of the principals behind the VRD dataset, for granting us permission to restore public availability of the images of the VRD dataset as part of our NeSy4VRD resource. We contributed NeSy4VRD to the computer vision, neurosymbolic and Semantic Web communities in 2023 to help foster more neurosymbolic research that leverages OWL-based knowledge graphs and OWL reasoning.

I also thank Tim Silman for choosing to base his research for his dissertation towards the degree of MSc Data Science, at City St George's, University of London, during the summer of 2024, on ideas associated with my notion of tensor knowledge graphs and their applications in neurosymbolic systems. As well as helping to demonstrate ideas, that collaboration helped to inspire my notion of a subsymbolic-symbolic neural network.

Abstract

A central theme of neurosymbolic AI is the idea of combining subsymbolic learning with symbolic reasoning, in some fashion, in order to get the best of both worlds. Our research examines this theme using OWL ontologies, OWL reasoning, and OWL-based knowledge graph tools to provide the symbolic reasoning. We explore combining subsymbolic learning with symbolic OWL reasoning in neurosymbolic systems. Our central research questions are: (i) how can we combine subsymbolic learning with symbolic OWL reasoning, and (ii) what are the effects or benefits of doing so?

Our research has three threads. Thread one presents NeSy4VRD, a unique dataset resource that facilitates neurosymbolic research, using OWL-based knowledge graph technologies, for visual relationship detection in images. NeSy4VRD combines an image dataset, and high-quality annotations, with a well-aligned, custom-designed, common sense OWL ontology, VRD-World. NeSy4VRD is small and extensible. It invites researchers to shape it to fit their research needs; to tailor its annotations, and to extend them, along with the companion ontology; and to share their extensions, so other researchers may compose enriched instances of NeSy4VRD.

Thread two of our research uses NeSy4VRD to explore ways of combining subsymbolic learning with symbolic OWL reasoning in neurosymbolic systems for detecting visual relationships in images. We present two investigations. Each one uses our OWL ontology, VRD-World, hosted in an OWL-based knowledge graph tool, together with OWL reasoning, in the guise of a symbolic reasoning engine. The two investigations exercise different aspects of OWL inference capability, and for different purposes.

Investigation 1 uses OWL ontologies (versions of VRD-World), and OWL reasoning, to logically augment the annotated scene graphs of the images in different ways, thereby altering the supervision they provide during neural network training. The results show how OWL reasoning can make a neural network's recall@N scores dance to an ontology's tune; how it can double precision, and more; how it can make subsymbolic learning faster. The results show the ontology matters: they show it controlling the OWL reasoning that augments the image scene graphs, and a neural network responding to the adjustments in the supervision it receives in training.

Investigation 2 of thread two uses our VRD-World OWL ontology, in a symbolic (OWL) reasoning engine, but this time in real-time, during neural network training, and during inference. In both settings, the role of the symbolic (OWL) reasoning engine is to act as a symbolic reasoning binary classifier, to classify visual relationships predicted by a neural network as either semantically valid or invalid. During training, the symbolic reasoning binary classifier helps to train a neural network (subsymbolic learning) classifier. It classifies emerging predictions. For those inferred to be semantically invalid, a semantic loss penalty is applied, in the manner of a logical constraint. This strategy reduces the relative frequency of semantically invalid predictions for

test set images dramatically. During inference, the symbolic reasoning binary classifier evaluates all of the candidate visual relationships predicted for test set images. The ones found to be semantically invalid are filtered-out, ensuring that only visual relationships that are semantically valid are submitted for performance evaluation. This strategy eliminates semantically invalid predictions entirely.

In thread three of our research, we consider the problem of combining subsymbolic learning with symbolic OWL reasoning from a different perspective. We present a novel way of representing the symbolic knowledge of an OWL-based knowledge graph, and a novel way of thinking about and performing OWL reasoning. And we blend these concepts with subsymbolic learning in neurosymbolic systems applications.

First, we focus on concepts. We introduce our notion of a tensor knowledge graph—a strictly binary, and hence strictly symbolic, tensor representation of an OWL-based knowledge graph. Then we introduce tensor knowledge graph reasoning—the emulation of many aspects of OWL reasoning using techniques based on known binary relational operations and implemented using matrix algebra and Boolean algebra.

Once these concepts are firmly established, we explore applying these concepts in neurosymbolic systems. We propose a novel neural network architecture that we call subsymbolic-symbolic, which wraps a subsymbolic learning module followed by a symbolic module. We describe an instance of this generic architecture we call a Combiner, which wraps a subsymbolic Classifier with a symbolic Generaliser. We inject knowledge—symbolic OWL knowledge encoded in a binary matrix, by the tensor knowledge graph reasoning engine—into the weight matrix of a Generaliser, and have it generalise classes predicted by the Classifier to the correct parent classes, using only the matrix algebra of a neural network forward-pass. We introduce our notion of 'neural symbolic inference', in which a symbolic Generaliser is equipped to perform actual symbolic inference, incrementally, within the rhythm of a standard neural network forward-pass. We show that, having been injected with partial knowledge of the VRD-World class hierarchy, a Generaliser can successfully infer its complete transitive closure, using an inference algorithm from the tensor knowledge graph reasoning engine, and adapted for the forward-pass setting of a neural network training cycle. We describe how our tensor knowledge graph is well-placed to find applications in the domain of knowledge graph embedding models, and we present a specification for a matrix factorisation of our tensor knowledge graph in order to facilitate these applications.

Keywords: *Artificial Intelligence, Neurosymbolic, Neural Networks, Subsymbolic Learning, Logic, Logical Constraints, Symbolic Reasoning, Semantic Web, OWL, Ontologies, Knowledge Graphs, Visual Relationship Detection, Scene Graph Generation, Tensor Knowledge Graphs, Tensor Knowledge Graph Reasoning.*

Contents

List of Figures **1**

List of Tables **5**

1 Introduction **7**
1.1 Overview 7
1.2 Background 12
1.2.1 Knowledge representation and reasoning (KRR) 12
1.2.2 The Web Ontology Language (OWL) 12
1.2.3 Knowledge Graphs 14
1.3 Research questions 15
1.4 Document structure 17

2 Related Work **18**
2.1 Neurosymbolic AI in general 18
2.2 Why OWL? 19
2.3 Thread one related work 23
2.3.1 Ontology engineering 23
2.4 Thread two related work 24
2.4.1 Visual relationship detection and scene graph generation 24
2.4.2 Neurosymbolic AI using loss-based logical constraints 25
2.4.3 Knowledge graphs in neurosymbolic systems 26
2.4.4 OWL reasoning in neurosymbolic systems 26
2.4.5 Symbolic reasoning engines in neurosymbolic systems 28
2.5 Thread three related work 28
2.5.1 Knowledge graph embedding models 29
2.5.2 Knowledge injection 32
2.5.3 Graph Neural Networks 32
2.5.4 Gunther Schmidt and relational mathematics 35

3 NeSy4VRD **38**
3.1 Overview . . . 39
3.2 The VRD dataset and visual relationships . . . 40
3.2.1 Attractive characteristics of the VRD dataset . . . 42
3.2.2 Unattractive characteristics of the VRD dataset . . . 43
3.3 NeSy4VRD annotated visual relationships . . . 46
3.4 Our OWL ontology design goals . . . 46
3.5 Overview of the VRD-World OWL ontology . . . 47
3.6 The VRD-World class hierarchy . . . 48
3.7 The VRD-World object properties . . . 51
3.8 Interpreting VRD-World . . . 53
3.9 NeSy4VRD support for extensibility . . . 54
3.9.1 Comprehensive code for dataset analysis . . . 54
3.9.2 The NeSy4VRD protocol . . . 55
3.9.3 The NeSy4VRD workflow . . . 56
3.9.4 Distributed annotation enhancement . . . 57
3.10 Summary . . . 59

4 Knowledge Graph Reasoning for Visual Relationship Detection **60**
4.1 The visual relationship detection problem . . . 61
4.2 Our baseline deep learning visual relationship detection system . . . 63
4.2.1 Overview . . . 63
4.2.2 Object detection . . . 63
4.2.3 Predicate prediction . . . 66
4.3 Metrics . . . 69
4.4 Knowledge graph reasoning for annotation augmentation . . . 72
4.4.1 Overview and methods . . . 72
4.4.2 The effects of OWL reasoning on recall@N (A) . . . 75
4.4.3 The effects of OWL reasoning on recall@N (B) . . . 77
4.4.4 The effects of OWL reasoning on the volume of predictions . . . 77
4.4.5 The effects of OWL reasoning on mAP@N . . . 78
4.4.6 The effects of OWL reasoning on hits per image . . . 81
4.4.7 The effects of OWL reasoning on learning speed . . . 83
4.4.8 The effects of OWL reasoning on symmetric pairs . . . 84
4.4.9 The effects of OWL reasoning on inverse pairs . . . 85
4.5 Knowledge graph reasoning for applying logical constraints . . . 87
4.5.1 Overview and methods . . . 87
4.5.2 The effects of OWL reasoning on semantic validity (A) . . . 92
4.5.3 The effects of OWL reasoning on semantic validity (B) . . . 93
4.5.4 The effects of OWL reasoning on recall@N . . . 94
4.6 Future work . . . 95

4.6.1 Proactive prediction evaluation . . . 95
4.7 Summary . . . 97

5 Tensor Knowledge Graphs **100**
5.1 The idea of an OWL-based tensor knowledge graph . . . 101
5.2 Our specification for an OWL-based tensor knowledge graph . . . 103
5.2.1 Our specification . . . 103
5.2.2 Current limitations . . . 105
5.2.3 Evidence that our tensor representation is accurate . . . 106
5.3 Tensor knowledge graph reasoning . . . 107
5.3.1 Key relational operations . . . 108
5.3.2 Emulating owl:inverseOf inference semantics . . . 109
5.3.3 Emulating rdfs:domain inference semantics . . . 115
5.3.4 RDFS and OWL inference rule coverage . . . 120
5.3.5 The Roy-Warshall transitive closure algorithm . . . 123
5.3.6 The Union-of-Powers transitive closure algorithm . . . 125
5.3.7 Union-of-Powers stopping conditions . . . 126
5.3.8 Comparing Roy-Warshall with Union-of-Powers . . . 130
5.4 Applications . . . 131
5.4.1 The Combiner: a subsymbolic-symbolic architecture . . . 131
5.4.2 Injecting symbolic class hierarchy knowledge . . . 133
5.4.3 Learning symbolic class hierarchy knowledge . . . 136
5.4.4 Neural symbolic inference . . . 138
The idea of neural symbolic inference . . . 138
Demonstrating neural symbolic inference using VRD-World . . . 140
Crossing the subsymbolic/symbolic divide differentiably . . . 142
5.4.5 Tensor knowledge graph embeddings . . . 144
5.5 Future work . . . 147
5.5.1 Handling OWL class descriptions . . . 147
5.5.2 Forward chaining . . . 148
5.5.3 Fast symbolic OWL reasoning for neurosymbolic systems . . . 148
5.5.4 Knowledge graph comparison services . . . 149
5.5.5 AI planning . . . 150
5.6 Summary . . . 151

6 Summary **154**
6.1 Thread one . . . 154
6.2 Thread two . . . 155
6.3 Thread three . . . 158
6.4 Closing remarks . . . 161

Bibliography **162**

A Our VRD-World Wikidata-inspired class hierarchy **173**

B Platform Benchmarking Results **177**

C Our mini VRD-World ontology **184**

List of Figures

1.1 The Semantic Web layer cake — a stack of open, standardised, integrated-by-design W3C technology specifications that together define the Semantic Web. 13

3.1 Two representative images from the VRD dataset with objects localised in bounding boxes and samples of representative annotated visual relationships rendered as user-friendly 3-tuples. 42

3.2 A partial pictorial rendering of the class hierarchy of the VRD-world OWL ontology (version 1). The object classes of the NeSy4VRD dataset are represented one-to-one by leaf classes in the hierarchy. To minimise clutter, the names of these leaf class are *enclosed within* the bubble that represents their main parent class. Lines represent sub-class/parent-class relationships. Classes coloured orange are new to NeSy4VRD and do not exist in the annotated visual relationships of the VRD dataset. All bubbles represent parent classes that were introduced to organise the NeSy4VRD leaf classes into a coherent hierarchy. To minimise clutter, this rendering of the VRD-World class hierarchy does *not* reflect *(i)* many subclass relationships, and *(ii)* many additional classes declared to be unions of other classes. 50

3.3 A partial pictorial rendering of the object properties of the VRD-world OWL ontology (version 1). Each NeSy4VRD visual relationship *predicate* is represented by a VRD-World object property coloured dark blue. Object properties that share semantics are gathered into bubbles that reflect their semantic category, coloured turquoise. Dotted lines represent sub-property relationships. Solid lines with double arrows represent semantic equivalence. To minimise clutter, this rendering does *not* attempt to express object property relationships such as inverses, or property characteristics such as symmetry and transitivity. 52

3.4 A zoom-in on the top-left portion of the VRD-world ontology class hierarchy shown in Figure 3.2. This zoom-in highlights classes `Vehicle` and `VehiclePart`, and the branches of subclasses beneath them. 53

4.1 A conceptual rendering of the visual relationship detection problem, and of the hypothesis that by combining symbolic OWL reasoning with deep learning, we can improve deep learning's predictive performance with respect to visual relationship detection in images. 62
4.2 Our baseline deep learning system for detecting visual relationships. An object detection neural network detects objects in images, and a multi-label predicate prediction neural network predicts relationships between ordered pairs of those objects. To study the predicate detection aspect of visual relationship detection in isolation, the noise of the object detection can be cancelled by using the objects annotated for the images in place of those detected by an object detector. 64
4.3 The setting for investigation 1 of thread two. The top half of the figure shows an OWL-based knowledge graph augmenting the annotated visual relationships of some hypothetical image. The presumption is that this knowledge graph hosts our VRD-World ontology. The bottom half of the figure shows the annotated visual relationships being used to provide supervision during PPNN training. A baseline deep learning system is trained using only the 'initial' sparse and arbitrary visual relationships for supervision. A neurosymbolic system is trained using the 'augmented' visual relationships that provide enriched supervision. 74
4.4 The effects of OWL reasoning on recall@N, in the predicate detection regime of experimentation (where the noise of object detection is cancelled), and when performance is evaluated against the baseline NeSy4VRD test set annotated visual relationships. . 76
4.5 The effects of OWL reasoning on recall@N, in the predicate detection regime of experimentation, when performance is evaluated against NeSy4VRD test set visual relationship annotations that have augmented by OWL reasoning using the same version of the VRD-World ontology as was used to augment the training set visual relationship annotations upon which each model is trained. 78
4.6 The effects of OWL reasoning on the distribution of the number of predicted visual relationships per image. 79
4.7 The effects of OWL reasoning on mAP@N. Different neurosymbolic treatments (combinations of OWL link inference semantics in OWL ontology VRD-World used to logically expand training image scene graphs) deliver mAP@N scores lower than a baseline deep learning system, to varying degrees. The salient observation, however, is how well mAP@N *endures*, given the corresponding, but far larger, relative increases in treatment prediction volumes induced by OWL reasoning. 80
4.8 Panel A: the effects of OWL reasoning on the mean number of predicted visual relationships per image that are actually submitted for performance evaluation. Panel B: the effects of OWL reasoning on the mean number of hits per image. 82
4.9 The effects of OWL reasoning on the speed of subsymbolic learning. 83

4.10 The setting for investigation 2 of thread two. The OWL-based knowledge graph hosting the VRD-World ontology, acting as a symbolic reasoning binary classifier, detects predictions emerging from the PPNN during training that are semantically invalid. A semantic loss penalty is applied to teach the predicate prediction neural network to not predict visual relationships that are semantically invalid. 87
4.11 An illustration of the OWL reasoning strategy used in investigation 2, where predictions of visual relationships are validated using symbolic (OWL) reasoning. The top panel shows the setting for PPNN training, the bottom panel for PPNN inference. In both settings, an OWL-based knowledge graph hosting OWL ontology VRD-World, acting as a real-time symbolic reasoning binary classifier, validates predictions of visual relationships. During PPNN training, predictions classified as semantically invalid (*e.g.*, `dog drive surfboard`) attract semantic loss penalties, as soft logical constraints. During PPNN inference, predictions classified as semantically invalid are filtered out, as hard logical constraints. 88
4.12 The effects of OWL reasoning, used as a logical constraint to suppress predictions of invalid visual relationships, on recall@N. 95
4.13 A vision of an OWL-based neurosymbolic visual relationship detection system that uses Datalog-like rules, executed in a rule engine, together with OWL-based knowledge graph reasoning and queries, to *proactively* scan the prediction space of a neural network, in order to identify logically plausible and logically implausible visual relationship predictions. Predictions of logically implausible visual relationships attract a logical constraint penalty, to discourage their prediction. Predictions that fail to emerge for logically plausible visual relationships attract a logical inducement penalty, to encourage the network to learn to predict these high-value (likely-to-be-a-hit) visual relationships. 96

5.1 A conceptual instance of a generic subsymbolic-symbolic neural network architecture, in which a subsymbolic learning module is followed by a symbolic module. In this instance of the generic architecture, the role of the subsymbolic module is filled by a Classifier that predicts classes, and the role of the symbolic module is filled by a Generaliser whose job is to generalise the classes predicted by the Classifier to their correct parent classes (per an OWL class hierarchy), but by strictly symbolic means only. The subsymbolic / symbolic divide is the point of transition between the subsymbolic and symbolic worlds. These three conceptual components are wrapped within a larger neural network module referred to as a Combiner. The subsymbolic learning module and the symbolic module work in-concert with one another, within the rhythm of a conventional neural network forward-pass. 132

5.2 Elements of an investigation into injecting symbolic knowledge encoding a class hierarchy into the Generaliser component of a subsymbolic-symbolic neural network architecture. The matrix on the left represents a binary matrix encoding of the class hierarchy on the right. The dotted links in the class hierarchy on the right are suggestive of the implied transitive closure of the hierarchy. The three fully-connected neural network layers in the middle represent the Classifier component of a subsymbolic-symbolic model. The single, larger layer of neurons to its right represents the Generaliser component of a subsymbolic-symbolic model. The binary matrix encoding the symbolic knowledge of the class hierarchy will ultimately be injected into the weight matrix of the single linear layer of the Generaliser, thereby enabling the Generaliser to find the parent classes of any base class predicted by the Classifier, using the normal matrix multiplication of a neural network forward-pass. 134

5.3 A setting for investigating learning symbolic class hierarchy knowledge. The binary matrix on the right is a symbolic encoding of a class hierarchy. The network on the left is a classifier being trained to learn that binary matrix, within the weight matrix of last layer of the network. A two-part loss function is employed for this purpose. A classification loss is computed based on the predictions of base classes, and generalisation loss is computed as a function of the difference between the weight matrix under study and the target binary matrix used for supervision. 137

A.1 A pictorial rendering of a potential alternate class hierarchy for the VRD-world OWL ontology, inspired by the publicly available Wikidata knowledge graph. Classes coloured blue represent Wikidata classes (entities, with their Wikidata 'Qnnnnn' numbers) to which a NeSy4VRD object class name has been assigned. All other classes have their Wikidata names. Classes coloured orange are duplicates introduced to avoid clutter from criss-crossing arrows. All arrows represent Wikidata `subClassOf` relationships. The colour yellow highlights classes with many subclasses and/or parents. 174

List of Tables

3.1 Image Datasets with Annotated Visual Relationships 43
3.2 Quantitative comparison of the NeSy4VRD and VRD annotated visual relationships (VRs). 46
3.3 Summary metrics for the VRD-World OWL ontology (v1) 48

4.1 The meaning of the names for the treatments used to explore the effects of OWL reasoning strategy 1 (annotation augmentation) on subsymbolic learning. For OWL reasoning strategy 1, treatments are configurations of activated link inference semantics within OWL ontology VRD-World, and the corresponding neural network (PPNN) models that emerge from having been trained on target scene graphs logically enriched by OWL reasoning governed by that configuration of VRD-World. 75
4.2 The effects of the annotation augmentation OWL reasoning strategy on precision, as measured manually via visual inspection of individual predicted visual relationships (VRs), for a small random sample of test set images. The adjusted precision scores exclude *false* false positives (FFPs) from consideration. The FFP rate gives the proportion of false positives deemed to be good predictions of visual relationships not annotated. Standard recall is hits per image over ground-truth annotations per image. . 81
4.3 Percentage changes in symmetric pairs relative to baseline for topN=50 84
4.4 Percentage changes in inverse pairs relative to baseline for topN=50 . . 86
4.5 Treatment definitions for OWL reasoning strategy 2 (prediction validation). 91
4.6 Comparison of semantic validity of predicted visual relationships for topN=999 92
4.7 Comparison of semantic validity of predicted visual relationship types for topN=999 94

5.1 Union-of-Powers Early-Stopping Analysis - For Binary Matrices of Size 500 127
5.2 Union-of-Powers Early-Stopping Analysis - Computational Savings per Matrix Size 129
5.3 Transitive closure algorithm runtime comparison 130

Chapter 1

Introduction

Our research explores ways of combining deep (subsymbolic) learning with symbolic, Web Ontology Language (OWL) reasoning in neurosymbolic systems. This chapter introduces our research and sets the scene for what is to come. We begin with a high-level overview that describes the three distinct but related threads of our research: 1) our novel 'image dataset plus OWL ontology' resource, NeSy4VRD, 2) our use of OWL-based knowledge graph reasoning for learning to detect visual relationships in (NeSy4VRD) images, and 3) our notion of *tensor knowledge graphs*—a novel matrix-based symbolic representation of OWL-based knowledge graphs, plus associated matrix-based symbolic reasoning techniques that emulate (aspects of) OWL reasoning. Following the overview, we provide background pertaining to the symbolic tradition of AI. Once ascendant, the symbolic tradition has been emerging from the shadow cast by the successes of the connectionist (neural network-based) tradition of AI. Semantic Web-based representatives of the symbolic tradition of AI (*e.g.*, OWL ontologies, OWL-based knowledge graphs, and OWL reasoning) permeate our work, and hence some appreciation of them is essential for understanding our neurosymbolic AI research and systems. After the background on Semantic Web-based concepts and technologies, we next specify and discuss our research questions. First we establish our most general research questions—the ones which span (are common to) the different threads of our research. Then we show how these specialise and multiply as they become thread-specific and experiment-specific. We close this introductory chapter with a description of the structure of the remainder of the document.

1.1 Overview

A central theme in neurosymbolic AI is the idea of combining subsymbolic learning with symbolic reasoning, in some fashion, in order to get the best of both worlds. Our

research examines this theme using OWL-based knowledge graphs and OWL reasoning to provide the symbolic reasoning. We explore combining subsymbolic learning with symbolic OWL reasoning in neurosymbolic systems. Our central research questions are: *(i)* how can we combine subsymbolic learning with symbolic OWL reasoning, and *(ii)* what are the effects or benefits of doing so? Our research has three threads. These threads are distinct but inter-related. They share our central research questions, but examine them differently.

Thread one of our research Thread one of our research into combining subsymbolic learning with symbolic OWL reasoning involves our creation of a unique dataset resource we call NeSy4VRD. The acronym NeSy4VRD stands for *Neurosymbolic AI for Visual Relationship Detection*. NeSy4VRD facilitates neurosymbolic systems research with OWL-based knowledge graph technologies, within the computer vision task of detecting visual relationships in images.

NeSy4VRD combines an image dataset, with high-quality annotations, and a well-aligned, custom-designed, common sense, companion OWL ontology. We call this common sense OWL ontology VRD-World. Resources like NeSy4VRD are scarce. We created its components in the process of pursuing our research vision. We packaged these components, wrapped them with documentation, and contributed NeSy4VRD to the neurosymbolic AI community in order to facilitate more neurosymbolic research using OWL-based knowledge graphs and OWL reasoning. NeSy4VRD is a neurosymbolic research enabler. It enables exploration of combining subsymbolic learning with symbolic OWL reasoning. NeSy4VRD enabled thread two of our research entirely, and its OWL ontology, VRD-World, helped to drive thread three. The contribution from thread one of our research is NeSy4VRD itself. Thread one of our research, NeSy4VRD, is covered in Chapter 3.

Thread two of our research In thread two of our research, we use NeSy4VRD to explore the problem of combining subsymbolic learning with symbolic OWL reasoning in the context of the computer vision task of detecting visual relationships in images. Thread two is our first opportunity to address our central research questions: *(i)* how can we combine subsymbolic learning with symbolic OWL reasoning, and *(ii)* what are the effects or benefits of doing so? We use an OWL-based knowledge graph tool hosting the OWL ontology, VRD-World, in the guise of a symbolic reasoning engine, to deliver OWL reasoning services that guide neural, subsymbolic learning. The close alignment between NeSy4VRD's VRD-World OWL ontology and the annotated visual relationships of its images ensures that the common sense symbolic reasoning that OWL performs is pertinent to the subsymbolic learning task.

We conducted two investigations for thread two. Both use a common, custom-designed

deep learning system for visual relationship detection. Using this purely subsymbolic system as a baseline, the two investigations explore different ways of introducing OWL reasoning, making the subsymbolic baseline neurosymbolic. The two investigations exercise different categories of OWL reasoning capability, and they employ different strategies for integrating OWL reasoning with deep learning. For evidence of the effects of OWL reasoning on subsymbolic learning, both investigations consider predictive performance, as measured by recall@N scores, and as revealed by analysis of the individual visual relationships predicted by the respective neurosymbolic systems.

Investigation 1 of thread two uses OWL reasoning to logically augment the annotated visual relationships (the scene graphs) of the images. The annotated scene graphs are the ground-truth targets that provide supervision to subsymbolic learning during system training. The hypothesis is that logically enriched supervision leads to better learning. Investigation 1 shows that the ontology matters. It shows a direct causal mechanism, from ontology through to system predictive performance. It shows an OWL ontology, VRD-World, controlling what OWL reasoning can and cannot do with respect to logically augmenting the image scene graphs. And it shows that the adjusted scene graphs provide adjusted supervision, which leads to adjusted subsymbolic learning, that registers as adjusted predictive performance. The adjustments can be dramatic, but not always. It depends on the ontology—on the particular mix of OWL inference semantics expressed in the ontology.

Investigation 2 of thread two of our research uses OWL reasoning to teach the deep learning system the semantics of the NeSy4VRD predicates, as reflected in the domain and range restrictions declared for them in the VRD-World OWL ontology. It uses OWL reasoning to teach subsymbolic learning to not predict visual relationships that, in VRD-World, are semantically invalid. The strategy involves applying a semantic loss penalty during training, in a manner akin to applying a logical constraint, whenever the deep learning system shows an inclination to predict something that OWL reasoning infers to be semantically invalid. The key to this strategy, once again, is the VRD-World OWL ontology. We designed it to enable OWL reasoning to enforce all of the domain and range restrictions declared within it.

Investigation 2 of thread two shows that an OWL-based knowledge graph tool hosting our VRD-World ontology can be used as a real-time symbolic reasoning engine to guide neural, subsymbolic learning during training. It demonstrates using this symbolic reasoning engine as a symbolic reasoning binary classifier to help train a neural network classifier. The symbolic classifier infers emerging predictions of visual relationships as being either semantically valid or semantically invalid. If it infers a prediction is invalid, a semantic loss penalty is computed and applied to steer the learning away from the prediction. Investigation 2 shows that this strategy is effective: the relative frequency of semantically invalid predictions on test set images drops dramatically.

Investigation 2 also demonstrates effective use of the real-time symbolic reasoning binary classifier during system inference on the test set. In this setting, the symbolic classifier is asked to evaluate every candidate predicted visual relationship for semantic validity, so that the semantically invalid ones can be filtered out, ensuring that only semantically valid ones are submitted for performance evaluation. This strategy eliminates invalid predictions entirely. This use case can be viewed as applying a logical constraint too, but a stronger one.

To close thread two of our research, we describe a further investigation that has been left for future work. The vision here is to extend OWL reasoning with Datalog-like rules, and to have a rule engine execute these rules, in co-operation with an OWL-based knowledge graph hosting VRD-World. The objective for the rule engine is to have it infer plausible visual relationship predictions for a given ordered pair of objects. If a neural network under training is not showing an inclination to predict a visual relationship that has been inferred to be plausible, then a plausibility penalty, or plausibility inducement (or prompting) can be applied, to prompt the network to learn the prediction that has been inferred to be plausible (*i.e.*, a likely hit). Thread two of our research is presented in Chapter 4.

Thread three of our research Thread three of our research explores combining subsymbolic learning with symbolic OWL reasoning from a perspective different to that of thread two. In thread three, we introduce new ways of thinking about how the symbolic knowledge of an OWL-based knowledge graph can be represented, and we introduce new ways of thinking about OWL reasoning, and of how it can be emulated using techniques that exploit the new representation. And we explore blending aspects of these symbolic concepts with subsymbolic learning in novel neurosymbolic network architectures. Thread three is covered in Chapter 5. First we introduce concepts; then we discuss applications these concepts in neurosymbolic systems. We close by discussing several categories of planned and potential future work relating to tensor knowledge graphs.

We begin by presenting our notion of a tensor knowledge graph—a strictly binary, and hence strictly symbolic, tensor representation of an OWL-based knowledge graph (or, equivalently, of an OWL ontology with accompanying data facts). Following this, we present our notion of tensor knowledge graph reasoning—the emulation of many aspects of OWL reasoning using basic binary relational operations on a tensor knowledge graph, implemented using matrix algebra and Boolean algebra. We show that our approach is not heuristic; it is well-grounded in a body of mathematical theory called *the calculus of relations*. We walk through specific examples showing how we emulate specific aspects of OWL inference semantics using our tensor knowledge graph reasoning techniques. We discuss our tensor knowledge graph reasoning engine, and point to its efficiencies

and scalability. We examine two binary matrix-based transitive closure algorithms used by our tensor knowledge graph reasoning engine, and describe early-stopping conditions we have identified that make one of them even more efficient. And we describe next steps in the development of tensor knowledge graphs, and of our tensor knowledge graph reasoning engine.

Once the concepts of tensor knowledge graph and tensor knowledge graph reasoning are firmly established, we shift the focus to discussing applications of these concepts in neurosymbolic systems. We begin by describing the target use case for the tensor knowledge graph reasoning engine, which is for it to be used as a real-time symbolic OWL reasoning engine in neurosymbolic systems. Then we introduce our concept of a neurosymbolic neural network architecture—an architecture we call subsymbolic-symbolic. We explain how this architecture wraps a subsymbolic learning module followed by a symbolic module, and where the two are separated by what we call the subsymbolic/symbolic divide. Then we describe an investigation involving an instance of this generic architecture that we call the Combiner, which wraps a subsymbolic Classifier with a symbolic Generaliser. The Classifier learns to predict base classes, and the Generaliser generalises these to their parent classes, per symbolic knowledge of an OWL ontology class hierarchy, encoded by a tensor knowledge graph, and injected into the symbolic Generaliser. The mechanism uses the matrix algebra of a conventional neural network forward-pass.

Next we discuss an investigation that extends the one just decribed. We present our notion of 'neural symbolic inference', in which the symbolic module of a subsymbolic-symbolic architecture, such as our Combiner, is equipped to do actual symbolic logical inference, incrementally, within the rhythm of a standard neural network forward-pass. We describe transferring one of the matrix-based transitive closure inference algorithms used in our tensor knowledge graph reasoning engine to a Generaliser, and how we adapted the inference algorithm for the setting of a neural network forward-pass. We demonstrate that, given injected symbolic knowledge of the VRD-World class hierarchy (as declared in the OWL ontology file), a Generaliser, thus equipped for inference, can correctly infer the full transitive closure of that class hierarchy.

Then we change tack and consider knowledge graph embeddings, and knowledge graph embedding models. We describe how our tensor knowledge graph is well-placed to find applications in the domain of knowledge graph embedding model research, particularly the category known as matrix factorisation models. And we point to the fact that our tensor knowledge graph reasoning engine may find applications here as well, given that some in the embeddings community advocate research that compares the efficacy of embeddings generated from non-materialised versus materialised knowledge graphs. We also specify a matrix factorisation for our tensor knowledge graph, to facilitate its use in knowledge graph embedding model research. We close part two of thread three of

our research by speculating about potential applications for the tensor knowledge graph reasoning engine in the field of AI planning.

1.2 Background

Here we provide background to our research. The background we cover pertains to the symbolic AI tradition. The symbolic AI tradition spans all three threads of our research. Thus, the topics we discuss here represent a common foundation that underlies and unifies our research.

1.2.1 Knowledge representation and reasoning (KRR)

Knowledge representation and reasoning (KRR) is a field of AI that sits within the symbolic tradition of AI. Its main concerns are *(i)* representing knowledge of the world in some machine-digestible form (often with the aid of some logic formalism), and *(ii)* exploiting the representation of the knowledge in order to reason over it, often using automated means (*e.g.*, Brachman and Levesque, 2004). Many roots of our research lead back to KRR.

1.2.2 The Web Ontology Language (OWL)

The Web Ontology Language (always abbreviated OWL) (*e.g.*, Allemang et al., 2020; Hitzler et al., 2012; Uschold, 2018) is a language expressly designed for representing knowledge. The form of the representation used by OWL is tailored for the context of the Semantic Web (Berners-Lee et al., 2001). As can be seen in Figure 1.1, OWL occupies a central position within the W3C open standards ecosystem of the Semantic Web (Allemang et al., 2020; Hitzler et al., 2010). Figure 1.1 is commonly referred to as the Semantic Web layer cake. Each layer corresponds to an open specification of a W3C Semantic Web standardised technology. Together, this set of integrated technology specifications is what defines the Semantic Web. In addition to OWL, two other layers in Figure 1.1 feature prominently in our research: those for RDFS and RDF. RDF (the Resource Description Format) defines the basic notion of an RDF triple—the basis of the Semantic Web's 'directed graph' approach to knowledge representation. RDF Schema (or RDFS) adds basic inference semantics to RDF, such as the notions of class and class hierarchy (Hays & Patel-Schneider, 2014). In our discussions, we sometimes refer to RDF and RDFS specifically. When we refer to OWL, we generally implicitly refer to RDF and RDFS as well, since OWL depends upon these companion technologies so heavily.

One active research area of logic-based KRR is Description Logic (Baader et al., 2007,

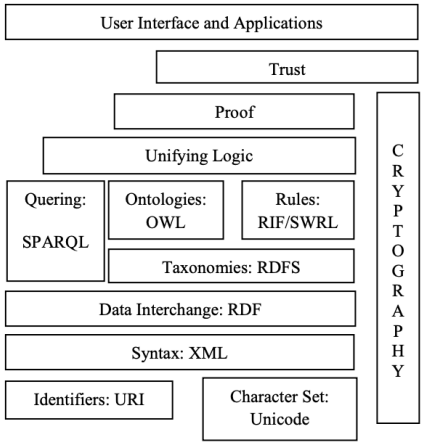


Figure 1.1: The Semantic Web layer cake — a stack of open, standardised, integrated-by-design W3C technology specifications that together define the Semantic Web.

2017). Per (Baader et al., 2017), Description Logic is built upon set theory and (binary) relation theory, where every *concept* is a set (of *individuals*), and every *role* (for relating individuals and concepts) is a binary relation. The area of Description Logic embraces a growing family of knowledge representation languages, each corresponding to a particular Description logic that supports a unique combination of inference semantics (or whose expressivity has a unique signature). Most Description logics are also decidable fragments of first-order logic (Krötzsch et al., 2013). OWL is, in fact, the Semantic Web incarnation of the expressive Description logic named $\mathcal{SROIQ}$ (Horrocks et al., 2006; Nardi & Brachman, 2003). The W3C's OWL has been described as perhaps the most important application of Description Logic (Krötzsch et al., 2013). It is through the Description logic formalism $\mathcal{SROIQ}$ that OWL is a *logic-based* knowledge representation language, and through $\mathcal{SROIQ}$ that our research has roots in, and is strongly related to, Description Logic.

An ontology expresses knowledge. It describes some domain of interest (Staab & Studer, 2009). An OWL ontology describes some domain of interest using OWL as its mode of expression. Since $\mathcal{SROIQ}$ is machine-digestible, an OWL ontology is machine-digestible. Public repositories of curated and reusable OWL ontologies exist, such as BioPortal (Whetzel et al., 2011) and OBO Foundry (Jackson et al., 2021) in the biomedical domain. Our research focuses on the use of custom-designed OWL ontologies, however. We rely on having the freedom to tailor our own OWL ontology in order to better explore leveraging discrete aspects of OWL reasoning for the special-purpose needs of our experiments.

The constructs of an OWL ontology have associated inference semantics (ontological inference rules). These permit OWL reasoning algorithms to reason over OWL on-

tologies, in the presence of data facts (expressed as RDF triples), in order to infer new knowledge (new RDF triples), and to enforce logical consistency constraints. Such reasoning typically takes place within the context of an OWL-based knowledge graph (described shortly). The reasoning services are typically managed by a particular OWL-based knowledge graph tool. An API enables interaction with the tool, and hence with the knowledge graph (the OWL ontology and data facts) it hosts, and with the OWL reasoning services it provides.

Since OWL is the Semantic Web version of $\mathcal{SROIQ}$, OWL reasoning is $\mathcal{SROIQ}$ reasoning. The central distinction between the two is that OWL reasoning exploits OWL's mode for representing knowledge, as expressed in an OWL ontology. Several efficient OWL reasoners exist, such as HermiT (Glimm et al., 2014), Pellet (Sirin et al., 2007), RDFox (Nenov et al., 2015), and ELK (Kazakov et al., 2011), that can reason over an OWL ontology, usually in the presence of accompanying symbolic *data* (facts asserting knowledge).

Given the preceding discussion of OWL, OWL ontologies and OWL reasoning, one can regard this package of Semantic Web (symbolic AI) technologies as something akin to a KRR system. OWL and OWL ontologies represent (symbolic) knowledge, OWL reasoning techniques (tailored for the RDF triple-based, or directed graph, representation) perform logical inference over the knowledge, and the underlying formalism is provided by the Description logic $\mathcal{SROIQ}$. This KRR system informs each of the three threads of our research. It informs the engineering of the NeSy4VRD OWL ontology, VRD-World, that is at the core of thread one. It is central to thread two's exploration of using OWL-based knowledge graph reasoning to guide and constrain deep (subsymbolic) learning in the task of detecting visual relationships in images. And it is fundamental to tensor knowledge graphs (as a knowledge representation scheme) and to tensor knowledge graph reasoning (techniques based on relational mathematics for performing logical inference over tensor knowledge graphs)—the topics central to thread three of our research.

1.2.3 Knowledge Graphs

Knowledge graphs are representations of symbolic knowledge that conform to a graph model, where nodes are concepts and entities of interest, and edges are relationships between them (Hogan et al., 2021). Knowledge graphs are fundamental to all of our research. Knowledge graphs were first popularised when Google announced its knowledge graph (Singhal, 2012). Since that time, interest in knowledge graphs has steadily grown. Several sources now provide broad coverage of all things relating to knowledge graphs, such as Chaudhri et al. (2022), Hogan et al. (2021), and Kejriwal et al. (2021). More recently, interest in knowledge graphs is being driven by explorations of their applications in neurosymbolic systems, particularly in connection with Generative AI,

as exemplified by large language models (LLMs), where knowledge graphs are used to help govern and tune LLM responses to questions (*e.g.*, Sequeda et al., 2025).

Not all knowledge graphs are Semantic Web knowledge graphs. For a knowledge graph to qualify as a Semantic Web knowledge graph, the graph's knowledge must be represented in a manner that adheres to the W3C specification for RDF (*i.e.*, axioms and facts must be expressed as RDF triples, the elements of which are Semantic Web URIs). A knowledge graph may (optionally) go further and embrace RDFS, or even OWL, as well, but support for RDF is sufficient to qualify as a Semantic Web knowledge graph. Examples of public Semantic Web knowledge graphs that support RDF, but not much more, include DBpedia (Lehmann et al., 2015), Wikidata (Vrandečić & Krötzsch, 2014) and Yago (Tanon et al., 2020). Such graphs can be huge in scale but, importantly, since the constructs of RDF do not carry associated inference semantics, Semantic Web knowledge graphs (like the ones just mentioned) that support RDF, but not RDFS or OWL, have no capability to reason over their contents in order to infer new knowledge.

Some Semantic Web knowledge graphs go beyond support for RDF and RDFS and embrace OWL as well. When we use the phrase 'OWL-based knowledge graphs', it is this category of Semantic Web knowledge graphs to which we refer. An OWL-based knowledge graph is governed by an OWL ontology that acts like a semantic schema for the graph. The ontology describes the domain of the knowledge graph by declaring (asserting) knowledge axioms. The inference semantics associated with the RDFS and OWL constructs used to declare the knowledge axioms determines what is (and is not) logically entailed by the facts (or data) in the graph. The ontology determines what OWL can and cannot do when it comes to reasoning over the knowledge graph.

In our research, we focus exclusively on OWL-based knowledge graphs. We do so because, amongst knowledge graphs, only OWL-based knowledge graphs possess the ability to reason over their contents—to infer new knowledge, and to catch logical inconsistencies. We do so because, by modifying an OWL ontology, we can modify the results of OWL reasoning. In other words, we manage OWL reasoning via an OWL ontology, to have it do what we want. Our research explores opportunities to manage and leverage OWL reasoning as we examine ways of integrating it with subsymbolic learning, in neurosymbolic systems—which is our subject of study.

1.3 Research questions

Our central research questions are: *(i)* how can we combine subsymbolic learning with symbolic OWL reasoning, and *(ii)* what are the effects or benefits of doing so? In Chapter 3, in which we discuss thread one of our research, NeSy4VRD, we cannot yet address our research questions, because NeSy4VRD is a neurosymbolic research enabler only.

Thread two of our research, in Chapter 4, is our first opportunity to address our research questions. In thread three of our research, in Chapter 5, we address our central research questions once again, but this time from a very different perspective than in thread two. This time, we address them from the perspective of integrating tensor knowledge graph concepts in novel subsymbolic-symbolic neural network architectures.

The settings of threads two and three of our research are so different that our central (rather general) research questions become specialised in very different ways. Here in this section, we concentrate on summarising how our central research questions become specialised, across threads two and three of our research, and across the different investigations conducted within these two threads of research.

Thread two research question specialisation In thread two of our research, in Chapter 4, we use symbolic OWL reasoning to guide subsymbolic learning in the context of the computer vision task of detecting visual relationships in images. Thread two involves two distinct investigations, each of which poses different questions.

In investigation 1 of thread two, our strategy for leveraging and integrating OWL reasoning with subsymbolic learning is to use OWL 'link inference' capabilities to augment (materialise) the annotated scene graphs of the training images. The hypothesis is that scene graphs enriched by OWL 'link inference' will provide better, more effective, and perhaps more efficient supervision during neural network training. Our research questions focus on looking for the effects of different mixtures of OWL 'link inference' on the visual relationship detection system's predictive performance. We ask, what are the effects of OWL reasoning on recall@N? We ask, what are the effects of OWL reasoning on the volume of predictions that the visual relationship detection systems generate? We ask, what are the effects of OWL reasoning on hits per image? We ask, what are the effects of OWL reasoning on the speed with which the subsymbolic learning takes place (as measured by the epoch numbers of the best performing models). For versions of our VRD-World ontology that possess inference semantics for symmetry, we ask: what are the effects of OWL reasoning on the system's ability to learn to predict in symmetric pairs? For versions of our VRD-World ontology that possess inference semantics for inverses, we ask: what are the effects of OWL reasoning on the system's ability to learn to predict in inverse pairs?

In investigation 2 of thread two, our strategy for leveraging and integrating OWL reasoning with subsymbolic learning is to use an OWL-based knowledge graph in the guise of a real-time symbolic reasoning binary classifier, so we know when to apply a semantic loss penalty as a logical constraint. The hypothesis is that this strategy will help to teach the system to not predict visual relationships that are semantically invalid. In this setting, our research questions focus on looking for the effects of OWL reasoning on the predictive performance of the visual relationship detection system, as measured by

the frequency of semantically invalid predictions. We ask, what are the effects of OWL reasoning on the frequency with which semantically invalid predictions are generated? We ask, what are the effects of OWL reasoning on the frequency with which semantically invalid visual relationship *types* are generated? And we ask, what are the effects of OWL reasoning on recall@N.

Thread three research question specialisation In thread three our research, in Chapter 5, we discuss tensor knowledge graphs and tensor knowledge graph reasoning, as well as their applications in neurosymbolic systems. In one application, we inject knowledge of a class hierarchy transitive closure, encoded by the tensor knowledge graph reasoning engine, into the weight matrix of a Generaliser module of a subsymbolic-symbolic network architecture. The hypothesis is that the Generaliser, equipped with injected knowledge, will correctly generalise any base class predicted by a Classifier to all of its parent classes (per an OWL ontology). We ask: can the Generaliser return the correct parent classes for a given base class predicted by a Classifier, by using nothing but the matrix multiplication of a standard forward-pass?

In another application investigation, we explore having a neural network learn the encoded binary knowledge just described, instead of injecting it. We ask, how well can subsymbolic learning learn binary-encoded symbolic knowledge, and having done its best at that task, how well can it generalise base classes to all of their parent classes?

In another application investigation, we explore the concept of 'neural symbolic inference' by equipping a Generaliser module to perform actual symbolic logical inference in order to infer the transitive closure of a class hierarchy on its own, incrementally, within the rhythm of a standard forward-pass. We ask, does this strategy work well enough that the Generaliser can correctly infer the transitive closure of the class hierarchy of our VRD-World OWL ontology.

1.4 Document structure

The structure of the remainder of this document is as follows. In Chapter 2, we discuss related work. In Chapter 3, we present NeSy4VRD, which represents thread one of our research. Chapter 4 presents thread two of our research, which explores the use of OWL-based knowledge graph reasoning for the purpose of improving predictive performance in the task of visual relationship detection in images. Chapter 5 presents thread three of our research: tensor knowledge graphs. We define the concepts of tensor knowledge graph and tensor knowledge graph reasoning, and then examine several applications of these concepts in novel subsymbolic-symbolic neural network architectures. In Chapter 6, we summarise the ground covered by our research, and reflect on its contributions.

Chapter 2

Related Work

Our research explores ways of combining deep (subsymbolic) learning with symbolic OWL reasoning in neurosymbolic systems. Recall that there are three distinct but related threads to this research: *(i)* NeSy4VRD, our 'image dataset plus ontology' resource, that facilitates neurosymbolic research with OWL-based knowledge graphs, *(ii)* using OWL-based knowledge graph reasoning for visual relationship detection in images, and *(iii)* emulating OWL-based knowledge graphs using tensor knowledge graphs and reasoning techniques based on relational mathematics, and exploring applications of these techniques in neurosymbolic systems. In this chapter, we present a systematic review of neurosymbolic (and other) literature relevant to our research. We begin by briefly acknowledging the broad neurosymbolic AI context within which all of our research takes place. Then, since OWL features so prominently in our research, we consider the question of 'why OWL?' by comparing it with other things, both inside and outside the Semantic Web, such as RDFS, propositional and first-order logic, and logic programming paradigms. Following this, we focus on each of the three threads of our research, in turn, by discussing thread-specific categories of related work.

2.1 Neurosymbolic AI in general

Our AI research is neurosymbolic. It sits within the landscape of the subfield of AI that has come to be known as neurosymbolic AI. The literature of neurosymbolic AI exhibits a vibrant diversity in approaches to blending the connectionist (subsymbolic) AI and symbolic AI traditions. Rather than attempt to summarise the broad range of these disparate approaches, we refer readers to helpful sources. Besold et al. (2017) provide a comprehensive survey of neurosymbolic learning and reasoning research prior to 2017. Sarker et al. (2021) and Hitzler and Sarker (2021) review more recent trends in neurosymbolic AI.

2.2 Why OWL?

In this section, we explain why OWL, and the Semantic Web, have proven to be good representatives of the symbolic tradition of AI for our neurosymbolic systems research. We do so largely by comparing them with other things. We begin by contrasting OWL with other aspects of the Semantic Web, *i.e.*, with RDF and RDFS. Then we consider OWL together with the Semantic Web as a whole, and in the process we highlight several advantageous features. Next we consider OWL by contrasting it with alternative logic-based knowledge representation languages outside the Semantic Web, namely propositional and first-order logic. We end by contrasting OWL (and OWL-based knowledge graph reasoning) with the logic programming paradigms Prolog, Answer Set Programming (ASP), and Datalog.

RDFS expressivity too limited Recall, from Figure 1.1, that OWL is a central component of the technology stack (or layer cake) of the Semantic Web—the suite of W3C specifications that enables knowledge graphs that are capable not just of representing knowledge, but of reasoning over it, deductively, as well. It is our commitment to using Semantic Web knowledge graphs in our neurosymbolic systems research that led us to embrace OWL and OWL-based knowledge graphs. As explained earlier, in Section 1.2.3, one can build Semantic Web knowledge graphs using RDF alone. But the focus of RDF is defining a foundation for expressing symbolic knowledge in a directed graph (RDF triple) format. Its expressivity is severely limited; and its constructs carry no inference semantics, so RDF permits no logical inference. RDFS adds important elements of expressivity by introducing classes and user-defined properties, and by permitting hierarchical (subsumptive) relationships to be defined between these (respectively). Several RDFS constructs carry inference semantics, and through these, the several inference rules defined for RDFS (Hays & Patel-Schneider, 2014) provide a solid foundation of logical inference capability—but a foundation only. Our decision to explore the application task of visual relationship detection in images, however, established requirements that far outstripped the knowledge expressivity of RDFS. To describe the domain of the NeSy4VRD dataset in a common sense way, and without making compromises, we needed not just RDFS but many aspects (albeit a small subset) of OWL's rich knowledge expressivity as well. The resulting OWL ontology, VRD-World, possesses rich inference semantics that permits extensive logical inference. Given our central research interest of combining symbolic reasoning with deep (subsymbolic) learning, this is just the sort of semantic schema (ontology) we desired for our knowledge graphs.

Open standards promote tool support Key advantages of the W3C's Semantic Web standards (of which OWL is one) are that *(i)* they are *specifications* (as opposed to implemented systems), and *(ii)* they are *open*. Together, these two features pro-

mote the development of reusable, often free, software tools and resources that help to make working with OWL and OWL-based knowledge graphs easy. Tool builders contribute and compete to provide superior implementations because they can be confident of supplying tools and services that seamlessly integrate and interoperate with all other implementations of the Semantic Web specifications. Researchers thus have a range of OWL-related tools from which to choose. For example, to design our OWL ontology, VRD-World, we used the freely available (and popular) OWL ontology editor named Protégé (Musen, 2015), developed by a team at Stanford University. Several efficient OWL reasoners have been developed, such as Pellet (Sirin et al., 2007), ELK (Kazakov et al., 2011), HermiT (Glimm et al., 2014), and RDFox (which is not open, but which has a free academic license) (Nenov et al., 2015). Pellet and HermiT (and other OWL reasoners) are also available as Protégé plug-ins. This allows ontology designers to readily integrate OWL reasoning into their ontology design process, making construction and refinement more efficient. We used both reasoners for this purpose, as well as a 3rd plug-in dedicated to identifying logical inconsistencies in OWL ontologies. Complete, full-service 'OWL-based knowledge graph management systems' that support the entire Semantic Web stack, and more, are also available, such as GraphDB (which is not open, but has a free version) ("Ontotext GraphDB", 2023) and RDFox. These and other advantageous features and benefits of OWL and OWL-based knowledge graphs (with respect to use in neurosymbolic systems) are discussed at greater length in Herron, Jiménez-Ruiz, and Weyde (2023).

Open standards promote learning The open standards of the Semantic Web also promote literature devoted to explaining its stack of technology specifications, especially OWL, which helps to promote learning and adoption of the Semantic Web generally, not just in AI. And since every source describes the same standards, the researcher (keen to learn quickly) finds welcome, reinforcing consistency across sources, which further promotes efficient learning.

OWL 2 profiles OWL 2 defines three *profiles* (Motik, Cuenca Grau, et al., 2012), each one corresponding to a subset of OWL 2 known to permit efficient processing in certain use cases or technology settings. The profiles place restrictions on the structure of OWL 2 ontologies. Each profile trades-off certain aspects of knowledge expressivity in order to prioritise computational efficiency. None of the profiles is a subset of another. The OWL 2 QL profile is optimised for scenarios involving huge amounts of instance data, and where query performance is of paramount importance, such as arise in Semantic Web models built upon relational database management systems. OWL 2 EL is optimised for scenarios involving huge ontologies, with very large numbers of classes and/or properties. OWL 2 RL is optimised for rule-based reasoning systems. By tuning OWL for particular settings, OWL 2 profiles help to ensure that OWL, OWL

reasoning and OWL-based knowledge graphs operate efficiently.

OWL vs propositional logic Now we highlight further features of OWL by discussing it in relation to alternative logic-based knowledge representation languages outside the Semantic Web. First, we consider propositional logic. Propositional logic (*e.g.*, Huth and Ryan (2004)) relies upon atomic proposition symbols with binary truth values, which can be combined using Boolean operations to form more complex formulae having binary truth values. One can arrange propositional symbols into groups and think of them as representing different categories of things, and craft formulae that express relations between these things (supposedly) of different categories. But to propositional logic, the symbols are logically indistinguishable.

The level of knowledge expressivity afforded by these restrictive features was insufficient for our needs. We aspired to represent the annotations of the NeSy4VRD dataset (*i.e.*, the scene graphs of visual relationships annotated for each image) in their entirety (training set or test set) in a knowledge graph. This required having the ability to represent the images themselves (their file names), the particular set of objects associated with each image, the classes of those objects, the integer coordinates of their bounding boxes, and the predicates relating ordered pairs of objects in individual visual relationships for each image. We further aspired to be able to then extract from the knowledge graph all of these same facts that had initially been asserted, and reconstitute the NeSy4VRD annotations exactly in their original format, and without loss of information. We achieved this objective, but to do so we needed a knowledge representation language like OWL whose expressivity embraces (and logically distinguishes between) objects (individuals), classes, (binary) predicates, and literals (*e.g.*, integers).

OWL vs first-order logic First-order logic (*e.g.*, Huth and Ryan (2004)) has the expressivity we needed to represent NeSy4VRD without loss of information. Recall, from Section 1.2.2, that OWL is the Semantic Web incarnation of the expressive Description logic $\mathcal{SROIQ}$, and that most Description logics (including $\mathcal{SROIQ}$) are decidable fragments of first-order logic. Hence OWL is, in fact, a subset of first-order logic—one that trades-off some expressivity in return for guaranteed decidability. Each of OWL 2's three profiles are particular decidable fragments of first-order logic, as well. While our OWL ontology, VRD-World, is non-trivial, it exercises only a small, commonly used subset of OWL's constructs (and, hence, inference semantics). So, for us at least, the opportunity cost of trading-off expressivity (in first-order logic) in return for guaranteed decidability (in OWL) was zero—a theoretical trade-off only. And knowing one cannot express something that becomes undecidable can be a welcome restriction.

OWL has another characteristic that distinguishes it from first-order logic in a significant

way: it can be easier to use. Since many of OWL's constructs have in-built (implicit) inference semantics (inference rules) associated with them, it is, in a sense, a *higher-level* language than pure first-order logic. In other words, with OWL, one can focus on expressing knowledge axioms that describe the domain of interest, and ignore the need for axioms that describe inference rules—since these are provided automatically, in the background, per the W3C specifications. Thus, it can be easier to describe non-trivial domains possessing rich inference semantics by using OWL than it would be using pure first-order logic. And OWL promotes consistency too, since the W3C standards reduce (if not eliminate) the risk of inconsistencies arising between the inference rules used by different researchers, making their results more comparable.

OWL vs logic programming paradigms Now we consider OWL in relation to various logic programming paradigms. The rationale for doing so rests on the observation that OWL can be used as much more than just a knowledge representation language for creating ontologies that describe domains. It is a component of the broader Semantic Web ecosystem of specifications and tools that enables provision of OWL-based knowledge graphs, together with integrated OWL reasoning services, and integrated knowledge graph query facilities (*e.g.*, via SPARQL, per Harris and Seaborne (2013)). As our research demonstrates, this package of technologies can be used to fashion symbolic reasoning engines for use in neurosymbolic systems, where an OWL ontology is used in a manner akin to a logic program—with which to reason over tiny worlds, to infer (and materialise) new knowledge, and to check of logical inconsistencies.

We consider OWL in relation to the logic programming paradigms of Prolog (*e.g.*, Bratko (2012)), Answer Set Programming (ASP) (*e.g.*, Gebser et al. (2012)), and Datalog (*e.g.*, Green et al. (2013)). These three paradigms share several characteristics. Each is declarative. Prolog and Datalog are both based on first-order logic. The syntax of ASP is rooted in first-order logic, but its semantics uses *stable model semantics*—a semantics designed by Gelfond and Lifschitz (1988) especially for use in logic programming. They all express programs (aka knowledge bases) consisting of facts and rules constructed using variants of Horn clauses. Each supports queries in relation to the facts and rules of a particular program (knowledge base). Each performs logical inference (albeit non-traditional in the case of ASP). Apart from the use of Horn clauses, and ASP's stable model semantics, at the right level of abstraction, OWL, OWL ontologies and OWL-based knowledge graphs can be said to share these same characteristics.

One weakness of Prolog is that, although declarative, the order in which the facts and rules appear in a program can (but not always) affect the logical inference outcomes of queries. OWL is immune to this problem. The synergies between OWL and Datalog are particularly strong, especially due to the OWL 2 RL profile, mentioned earlier, which is designed for use in rule-based reasoning technologies, of which Datalog is an exemplar.

The synergies are so strong here that advanced OWL-based knowledge graph tools like RDFox convert an OWL ontology, and any RDF triple-based facts, into Datalog, so that all reasoning can be performed using Datalog reasoning techniques. This strategy has the secondary benefit of permitting seamless integration with user-defined Datalog rules that one might create in order to extend OWL's expressivity and/or logical inference capabilities. This topic is discussed more fully in Herron et al. (2025).

2.3 Thread one related work

Here we discuss literature that relates to thread one of our research: the creation of our NeSy4VRD resource that facilitates neurosymbolic systems research with OWL-based knowledge graphs, especially with respect to the computer vision task of detecting visual relationships in images.

2.3.1 Ontology engineering

In thread one of our research, NeSy4VRD, the requirement was to create a resource consisting of an image dataset, with high-quality annotations, and a well-aligned, companion OWL ontology. We wanted the annotations and the ontology to be well-aligned so that any OWL reasoning that might be performed would yield results that were directly pertinent to the subsymbolic learning task of detecting visual relationships in the images of the dataset. Designing an appropriate OWL ontology was the primary intellectual challenge implied by our requirements. Thus, the literature and tools most relevant to thread have to do with ontology engineering.

There is a large body of literature covering the general topic of ontology engineering (*e.g.*, Allemang et al., 2020; Keet, 2020; Kendall and McGuinness, 2019; Noy and McGuinness, 2001). We took guidance from these sources. Keet (2020) distinguishes between 'bottom up' and 'top down' approaches to engineering and ontology. We adopted the 'bottom up' approach. We used the object classes of the NeSy4VRD annotated visual relationships as leaf classes in our OWL class hierarchy, and worked our way upwards to create a reasonable, common sense hierarchy. We used the predicates of the NeSy4VRD annotated visual relationships as OWL user-defined object properties, and then declared each one to have appropriate characteristics and relationships, such as symmetry, transitivity, inverses, equivalence relationships, and subproperty relationships.

The formalised the specification of our VRD-World ontology using the free ontology editor Protégé (Musen, 2015). We took advantage of free Protégé plug-in utilities designed to support ontology development, such as ontology debuggers. Many machine

learning tools exist to support various different aspects of ontology development such as, for example, concept learners (*e.g.*, d'Amato, 2020).

2.4 Thread two related work

Here we discuss neurosymbolic systems research that relates to thread two of our research: leveraging symbolic OWL reasoning to guide and improve neural, subsymbolic learning within the context of the computer vision task of detecting visual relationships in images.

2.4.1 Visual relationship detection and scene graph generation

Our NeSy4VRD resource, and therefore much of our research, is based on the VRD dataset, introduced by Lu et al. (2016). This paper does not mention anything to do with neurosymbolic AI, but from today's perspective it clearly is neurosymbolic research. Their visual relationship detection system exploits word embeddings of the object class names and predicate names in the VRD dataset.

Donadello and Serafini (2019) explore visual relationship detection on the VRD dataset using Logic Tensor Networks (LTN) (Badreddine et al., 2022; Serafini & d'Avila Garcez, 2016). Their strategy was to use LTN *Real Logic* knowledge axioms to declare *negative* domain and range constraints for the predicates of the VRD dataset. The strategy they employed inspired the investigation we describe in Section 4.5, where we use an OWL-based knowledge graph hosting our VRD-World ontology as a symbolic reasoning binary classifier, to teach our system to not predict semantically invalid visual relationships. In other words, to teach our system the domain and range restrictions of the user-defined properties declared in our VRD-World ontology.

The literature concerning visual relationship detection has two communities that use different language to talk about (what amounts to, in our view) the same thing. One community, perhaps inspired by the original VRD paper (Lu et al., 2016), prefer the language of visual relationships and visual relationship detection. The other community prefers to speak of image scene graphs, and of scene graph generation. The survey by Zhu et al. (2022) makes this clear. It shows that the VRD dataset itself has been a popular choice in scene graph generation research. We see a collection of visual relationships for an image as equivalent to a scene graph for an image. We use a bit of both forms of language. As explained in Khan et al., 2022; Zhu et al., 2022, scene graphs (and therefore collections of visual relationships too) are commonly used as precursors to a variety of enriched, downstream visual understanding and reasoning application tasks,

such as image captioning, visual question answering, image retrieval, image generation and multimedia event processing.

2.4.2 Neurosymbolic AI using loss-based logical constraints

One of our investigations into using OWL reasoning to influence subsymbolic learning employs a strategy of applying logical constraints via the loss function of a neural network under training. This investigation is described in Section 4.5. We use an OWL-based knowledge graph hosting our VRD-World ontology as a real-time symbolic reasoning engine. It functions like a symbolic reasoning binary classifier. The classification decisions it makes, regarding the semantic validity or invalidity of a predicted visual relationship, are used to apply a semantic loss penalty to help guide the subsymbolic learning. Our approach is one variation of a common theme, or pattern.

Others have explored different variations of the same theme. LTN functions entirely on the basis of applying logical constraints via the loss function. Each *Real Logic* knowledge axiom represents a discrete constraint. When LTN is used in the default manner, the loss function is nothing but the set of logical constraints (axioms) that have been declared. The strategy is for the neural network under training to learn to best satisfy the set of constraints.

Another variation on the theme of applying logical constraints via the loss function is seen in research that uses the ROAD-R dataset (Giunchiglia et al., 2023). The ROAD-R dataset contains autonomous vehicle driving videos whose frames (images) have been annotated with the objects they contain. The dataset supports visual relationship detection-type applications. A set of 243 propositional logic 'requirements' has been manually specified to accompany the dataset. The requirements define the permissible combinations of labels for 10 agent classes, 19 agent action classes, and 12 agent location classes. These requirements are used as logical constraints that must be satisfied. If the network under training shows an inclination to predict something, a reasoner does a pass over the set of requirements to see if any are not satisfied. If some are not satisfied, a loss penalty is computed and applied. There are strong parallels between research that involves ROAD-R and investigation 2 of thread two of our research, presented in Section 4.5.

We think it may well be feasible to design an OWL ontology (one or more, each of which explores a different ontology design strategy) that emulates and enforces the set of propositional logic requirements specified for the ROAD-R dataset. Were this to be achieved, one can imagine using an OWL-based knowledge graph tool to host that OWL ontology, in the guise of a real-time symbolic reasoning engine, to perform and replicate the ROAD-R experiments.

2.4.3 Knowledge graphs in neurosymbolic systems

As interest in neurosymbolic AI research has grown, so has the frequency with which knowledge graphs feature as symbolic components in neurosymbolic systems (Hitzler, 2021). One example of this is the progressively developing theme of 'deep deductive reasoning' (Bianchi & Hitzler, 2019; Ebrahimi, Eberhart, et al., 2021; Ebrahimi, Sarker, Bianchi, et al., 2021), where neural networks are trained to reason over knowledge graphs.

Knowledge graphs have also been shown to be helpful when data samples are expensive, difficult or impossible to obtain, such that there is a lack of data with which to train robust deep learning-based systems, as in few-shot and zero-shot learning scenarios (*e.g.*, J. Chen, Geng, et al., 2021; Z. Chen, Chen, et al., 2021; Geng et al., 2021).

2.4.4 OWL reasoning in neurosymbolic systems

Research that uses OWL reasoning in neurosymbolic systems is scarce. Breit et al. (2023) conducted a systematic mapping study of 476 recent research papers that explore combining Semantic Web technologies with machine learning in some way. They report that only 29 (about 6%) of these papers mention using semantic processing modules of some kind (where, by 'semantic', they mean symbolic knowledge representation). The dominant use cases for these modules relate to rule sets (learning them, improving them), and to data enrichment. The study also finds that of these 29 papers, only 20 (about 4% of the total) mention using reasoning capabilities to infer knowledge.

The data and the analyses of this systematic mapping study are publicly available in a companion knowledge graph. The authors call this the Semantic Web and Machine Learning Systems knowledge graph, or SWeMLS-KG (Ekaputra et al., 2023). We queried this knowledge graph to find the 20 papers that discuss using some form of Semantic Web reasoning capability. We found 17. Of these, we found only 5 that use OWL reasoning in some way.

Another recent study and vision paper, d'Amato et al. (2023), comes to similar conclusions regarding the scarcity of research that leverages Semantic Web symbolic reasoning. The authors review the role of knowledge graphs generally (not just Semantic Web knowledge graphs) in machine learning. They point to gaps and opportunities. But they also observe that knowledge graph symbolic reasoning methods are under-explored and largely disregarded.

Research that leverages OWL reasoning in neurosymbolic systems may be scarce, but it exhibits ingenuity. In the paragraphs that follow, we describe four of the five papers we found in the SWeMLS-KG referred to above.

Chang et al. (2020) exploit OWL 'type' inference capability—the ability for OWL reasoning to infer the classes to which an individual belongs. The authors describe a tutoring system that can react intelligently in response to interactions with human learners. A custom OWL ontology models the tutoring system domain and contains descriptions of classes that correspond to tutoring system actions. Data regarding learner interactions with the system are progressively loaded into the system's knowledge graph. The OWL reasoner HermiT reasons over the data in the knowledge graph to infer new knowledge based on the learner interaction data. In the process, each learner interaction is classified as belonging to one of the tutoring action classes, resulting in inferred triples such as, say, `(learnerX-interactionN rdf:type GiveEncouragement)`. Such inferred triples are interpreted as predictions (instructions) of the best next action for the tutoring system to take. The mechanism works because OWL allows a class (a domain concept) to be described in terms of the characteristics that must be possessed by individuals in order for them to be members of the class. When OWL reasoning detects that an individual possesses the right mix of characteristics, it infers that the individual is a member of the appropriate class, and materialises (makes explicit) this inference with new knowledge: a new triple in the knowledge graph. We leverage OWL 'type' inference capability in one of the investigations in thread two of our research.

Donadello et al. (2019) present a digital healthcare neurosymbolic system. They use a custom OWL ontology to describe dietary and physical activity domains, and healthy lifestyle behaviours. They supplement their custom OWL ontology with SPARQL rules—SPARQL being the Semantic Web's knowledge graph query language (Harris & Seaborne, 2013), which is flexible enough to also be exercised in the context of rules. The supplementary SPARQL rules model unhealthy lifestyle behaviours. User diet and activity data are loaded into the system's knowledge graph. The RDFpro tool (Corcoglioniti et al., 2015) drives the reasoning and the rule execution. If data in the knowledge graph indicate that a SPARQL rule for an unhealthy behaviour has been *satisfied*, the rule itself infers a new triple into the knowledge graph to signal an instance of the unhealthy behaviour. These triples (signalling unhealthy behaviours) are then rendered into natural language to encourage healthier user behaviours.

Mouakher et al. (2019) use a custom OWL ontology in a system for monitoring the bio-health of vineyards. They supplement their OWL ontology with SWRL rules—SWRL being the Semantic Web Rule Language (Horrocks et al., 2004). A wireless sensor network surrounding the vineyard measures micro-climate conditions. These data are fed into the system's knowledge graph. The OWL reasoner Pellet reasons over the knowledge graph to infer new triples that are interpreted as predictions of risk of impending diseases and pest infestation.

Nakawala et al. (2019) use a custom OWL ontology supplemented with SWRL rules as part of a neurosymbolic system for recognising surgical processes for robot-assisted

surgery. A convolutional neural network (CNN) recognises the *current* step of a surgical workflow. A recurrent neural network (RNN) predicts the *next* step of the surgical workflow. And by reasoning over these inputs in the presence of the OWL ontology and the SWRL rules, the OWL reasoner Pellet infers supplementary surgical context information, such as the surgical phase, the surgical instruments used, and actions to be taken.

2.4.5 Symbolic reasoning engines in neurosymbolic systems

The neurosymbolic system AlphaGeometry (Trinh et al., 2024) combines a large language model (LLM) with a symbolic deduction engine. The deduction engine uses Horn clause geometry rules, algebra rules, and associated inference algorithms. The LLM and the symbolic deduction engine co-operate in solving geometry problems.

A subsymbolic learning system (of some kind) co-operating with a symbolic reasoning system (of some kind) is a particular *pattern* of neurosymbolic system architecture (per, *e.g.*, Breit et al., 2023; van Bekkum et al., 2021; van Harmelen and ten Teije, 2019). AlphaGeometry is one instance of this pattern. Systems built for the ROAD-R dataset (Giunchiglia et al., 2023), where propositional logical inference enables learning to be guided via the application of logical constraints, are instances of this pattern.

Much of our research is best viewed through the lens of this pattern. Its most central theme involves using OWL ontologies, OWL reasoning, and OWL-based knowledge graph tools in the guise of symbolic reasoning engines in neurosymbolic systems. This vision motivated the creation of our VRD-World OWL ontology—an exercise which ultimately led to NeSy4VRD as a whole (thread one of our research). In thread two, we employ this neurosymbolic system architecture pattern by using symbolic (OWL) reasoning engines to help neural networks detect visual relationships in images. And this architectural pattern also helped to motivate our development of tensor knowledge graphs, tensor knowledge graph reasoning, and our emerging tensor knowledge graph reasoning engine (per thread three of our research).

2.5 Thread three related work

Here we discuss research and mathematical theory that relates to thread three of our research: tensor knowledge graphs and tensor knowledge graph reasoning that emulates aspects of OWL reasoning.

2.5.1 Knowledge graph embedding models

Knowledge graphs (in general) have inspired much neurosymbolic research into encoding the symbolic background knowledge they contain in real-valued vectors. These vector are commonly referred to as *knowledge graph embeddings*, and the models that give rise to them as *knowledge graph embedding models*. The general objective is for the embeddings to preserve semantic similarity, and to reflect this similarity by proximity within the vector space of the embedding model. This enables forms of geometric inference to be explored. A popular application for knowledge graph embeddings is *knowledge graph completion*, where techniques are employed to predict new triples deemed to be missing from the graph and which, if added, would help to complete the graph. See, for example: Z. Chen et al. (2020), Dai et al. (2020), d'Amato (2020), Nickel et al. (2015), Rossi et al. (2021), and Wang et al. (2021). Like all knowledge graphs, OWL-based knowledge graphs are readily used for generating knowledge graph embeddings for applications in neurosymbolic systems. OWL2Vec* (J. Chen et al., 2021) is one embedding model designed specifically for use with OWL-based knowledge graphs.

Matrix factorisation embedding models The survey by Rossi et al. (2021) proposes a taxonomy of knowledge graph embedding models with three categories: matrix factorisation (tensor decomposition) models, geometric models, and deep learning models. The matrix factorisation models begin by representing a knowledge graph as a single 3D binary tensor which is then factorised (decomposed) in some way using linear algebra techniques. The matrices produced by the factorisation contain real-valued vectors that can be used as embeddings for the entities and relations of the knowledge graph in downstream applications. The decomposition can also be used for knowledge graph completion purposes. When the decomposition is used to reconstruct the original 3D binary tensor (using matrix multiplication), the hypothesis is that any newly appearing 1s (or near 1s) in the cells of the reconstructed tensor can be interpreted as representing predictions of triples that are missing from the graph. Most of the matrix factorisation models mentioned by Rossi et al. (2021) turn out to be variations of the *RESCAL* knowledge graph embedding model, introduced by Nickel et al. (2011), although, for some reason, RESCAL is not mentioned in the survey.

The single 3D binary tensor representations used by matrix factorisation knowledge graph embedding models are intended for simple (general) knowledge graphs—graphs consisting only of data triples that relate individuals to each other. These simple models correspond to just one of the five 3D binary tensors that we use to represent an OWL-based knowledge graph in a tensor knowledge graph (see Chapter 5). Although we conceived our notion of an OWL-based tensor knowledge graph independently of these matrix factorisation knowledge graph embedding models, their starting point—a 3D binary representation of a knowledge graph—lends welcome support to our notion. The

simplicity of the representation also helps to highlight our more ambitious intentions for our five-tensor binary representation of a knowledge graph. The single 3D binary tensor representation is designed to produce embeddings in a one-time decomposition exercise; and, optionally, to produce predictions to help complete a knowledge graph in a one-time reconstruction exercise. Our objective for tensor knowledge graphs, on the other hand, is to represent as much of the expressiveness of an OWL-based knowledge graph as feasible, so as to enable OWL reasoning to be emulated as far as feasible, using logical inference techniques based on matrix algebra and Boolean algebra. In spite of this difference in objectives, in Chapter 5 we specify a RESCAL-inspired matrix factorisation scheme for our 5-3D-binary-tensor tensor knowledge graph model. We do this to facilitate exploration of our model's utility as a source for matrix factorisation-based knowledge graph embeddings, and for matrix factorisation-based link prediction for knowledge graph completion.

Geometric embedding models It may be that our 5-3D-binary-tensor tensor knowledge graph model can also find applications within categories of knowledge graph embedding models other than the matrix factorisation category just discussed. For this reason, we briefly review some other categories of knowledge graph embedding models. We begin by describing examples of what Rossi et al. (2021) call *geometric* knowledge graph embedding models. The first of these was TransE (Bordes et al., 2013). TransE embeds the entities and relationships of knowledge graphs based upon the central idea of interpreting relationships as (vector space) translations that operate on the embeddings of entities. Its basic assumption is that, if a triple `(h, r, t)` exists, then the following expression involving the three corresponding embedding vectors should hold: $\mathbf{h}+\mathbf{r} \approx \mathbf{t}$. Once again, the main application is knowledge graph completion by predicting (supposedly missing) links between entities. According to Rossi et al. (2021), however, TransE is unable to model symmetric relations, transitive relations, one-to-many relations and many-to-one relations.

Successor models explore variations of TransE, each designed to address limitations such as the ones just listed. One of these is RotatE (Sun et al., 2019), which interprets relationships as rotations of entity embeddings, and in a complex (rather then real) vector space. In RotatE, the basic assumption is that, if a triple `(h, r, t)` exists, then the following expression involving the three corresponding embedding vectors should hold: $\mathbf{h} \circ \mathbf{r} \approx \mathbf{t}$, where $\circ$ is the Hadamard (or element-wise) product. According to Rossi et al. (2021), RotatE successfully models relation symmetry, anti-symmetry, inversion, and composition.

Box embedding models Another category of knowledge graph embedding models has come to be called *box* embedding models. The model BoxE (Abboud et al., 2020) embeds entities as points (vectors) and relations as sets of *axis-aligned hyper-*

rectangles (regions of hyper-space with orthogonal boundaries, loosely referred to as *boxes*). An n-ary relation is represented by a set of n boxes. In hyper-space, the set of boxes for a relation may or may not intersect with and/or be subsumed by one another. A binary relation is represented by two boxes: possibly disjoint, possibly intersecting to some degree, and possibly one is fully subsumed within the other. The relations between the boxes in hyper-space have been found to encode relevant logical properties. For example, a large box subsuming a small box models a hierarchical relationship. Intersection models conjunction. Disjoint boxes model mutual exclusivity. Boxes model sets, and entities (point vectors) contained within boxes model set membership. Because the point and box embeddings capture these logical properties, they can also be used for aspects of logical inference, and for link prediction, to predict (supposedly missing) facts. These same concepts and logical inference opportunities are leveraged by Ren et al. (2020) in their query answering system, Query2Box. With Query2Box, logical queries, including ones involving conjunction, disjunction, and existential quantification, are themselves embedded as sets of axis-aligned hyper-rectangles (boxes). Points (entities) found to be inside query boxes are interpreted as answer sets—the entities that answer queries.

Other embedding models OWL2Vec* (J. Chen et al., 2021) is yet another knowledge graph embedding model. The embedding models discussed so far presume knowledge graphs that consist of entities and relations only. OWL-based knowledge graphs, on the other hand, distinguish between three categories of things: individuals (objects), classes, and (binary) relations. OWL2Vec* belongs to a category of knowledge graph embedding model conceived to generate embeddings specifically for OWL-based knowledge graphs governed by an OWL ontology, in the hope of capturing the rich logical relationships expressible in such knowledge graphs.

Under the covers, OWL2Vec* relies on Word2Vec (Mikolov et al., 2013) word embeddings, where the words are URIs drawn from an OWL-based knowledge graph (or OWL ontology). The context windows for skip-gram Word2Vec are generated by doing repeated random walks (of a certain length) of the knowledge graph. Word2Vec learns embedding vectors using a shallow neural network possessing just two layers: one hidden layer (with no activation function) and one output layer. The embedding vectors learned by the neural network are actually the rows of the weight matrix of the hidden layer. In skip-gram mode, the Word2Vec network is trained on word pairs, (input word, target word), with corresponding binary labels indicating whether the word pair represents a positive or negative example. The Word2Vec network is thus trained as a binary classifier and, for a given input word, it learns a distribution of probability over the words in the vocabulary that appear nearby the input word within the training corpus. But the probability distributions learned by the network are not of interest; only the weights of the trained network are harvested, as embedding vectors.

2.5.2 Knowledge injection

Another popular use case for embeddings of knowledge graph symbolic knowledge in neurosymbolic systems is *knowledge injection*. As explained by Buffelli and Tsamoura (2023), the phrase 'knowledge injection' is interpreted broadly. It can be used to refer to the injection of knowledge in symbolic form, as in Logic Tensor Networks (Badreddine et al., 2022; Serafini & d'Avila Garcez, 2016), for example, where fuzzy-logic knowledge axioms are woven into loss functions. Frequently, however, the phrase refers to the injection of knowledge represented in subsymbolic form, *i.e.*, as embedding vectors (*e.g.*, Fu et al., 2023). Sheth et al. (2019) refer to the injection of subsymbolic (embedded) knowledge as yielding *knowledge-infused learning*. One subcategory of subsymbolic knowledge injection involves the use of knowledge graph embeddings as domain knowledge supplements to primary training data (*e.g.*, Myklebust et al., 2022). The hypothesis here is that 'data + knowledge' can enhance deep learning.

Some of the investigations we present in thread three of our research, in Chapter 5, explore the theme of knowledge injection. The perspective from which we consider knowledge injection is that of our notion of a tensor knowledge graph; and the knowledge we inject into neural networks is strictly symbolic knowledge, encoded by, and extracted from, a tensor knowledge graph. One category of symbolic knowledge encoded within a tensor knowledge graph is that of the class hierarchy of an OWL ontology. We discuss injecting the binary matrix that encodes this symbolic knowledge of a class hierarchy directly into the weight matrix of a 'symbolic' neural network layer, thus enabling it to generalise predictions of base classes to their parent classes, using standard neural network forward-pass matrix multiplication operations.

2.5.3 Graph Neural Networks

Here we discuss Graph Neural Networks (GNNs), a family of neural network architecture designed for processing data represented as graphs (collections of nodes and edges). Given the centrality of OWL-based knowledge graphs to our research, it is appropriate to discuss GNNs in order to characterise them in relation to our neurosymbolic research, and thus to help explain *(i)* where they might conceivably have featured in our research, and *(ii)* why, ultimately, they do not. As will become clear, the strongest affinity between GNNs and our neurosymbolic research relates to thread two and the computer vision task of detecting visual relationships in images. We discuss GNNs here, however, as part of thread three related work, because of the strong relationship that exists between them and knowledge graph embedding models, which we discussed just above.

Several surveys exist that describe GNNs themselves, and the GNN landscape, such as Wu et al. (2021), Zhang et al. (2022), and Zhou et al. (2020). GNNs can be viewed

as generalisations of neural networks designed for processing flat feature vector data, sequences, and images to networks that can handle arbitrary graph-structured data. The general goal of GNNs is to learn embedding vector representations for nodes, edges, subgraphs, or entire graphs, such that the representations are useful for various graph-related prediction tasks. Hence, the ambition is for the embeddings to capture graph structure/connectivity (local graph context) and encode available features of the graph's neighbourhood elements.

The original GNN model, designed for simple homogeneous graphs (those having one type of node, and one type of edge), was proposed by Scarselli et al. (2009). The central idea is that of learning the embedding vector for node i from the embeddings for the nodes in the neighbourhood of node i, $\mathcal{N}(i)$. For this purpose, a node embedding update function aggregates the embeddings for nodes in $\mathcal{N}(i)$. This original GNN model is node-centric; it does not embed edges (relations). But its node embedding update function does allow edge features (*e.g.*, edge labels) to participate and thereby influence the embeddings learned for the nodes. So, to some degree at least, the model can be said to already be edge-aware.

The Scarselli et al. (2009) GNN model inspired a series of reformulations aimed at extending and refining its capabilities. During this evolution, the concept of learning node embeddings by iterating over and aggregating neighbouring node embeddings became formalised and known as *message passing*. Gated Graph Neural Networks (GGNNs) (Li et al., 2016) introduce a GRU (Gated Recurrent Unit, per Recurrent Neural Networks) into the node embedding update function. The gating mechanism controls how much new information from the neighbourhood gets integrated into each node embedding update, and how much of the current embedding vector (state) persists. GGNNs are node-centric: they do not embed edges (relations). Limited support for multi-relational graphs is available in the node embedding update function, via a matrix encoding the connectivity (edges) of the graph. This permits edge-specific (and direction-specific) parameters to be supplied that can influence the embeddings learned for the nodes.

Graph Convolutional Networks (GCNs) (Kipf & Welling, 2016) generalise the convolution operation to graphs (in a manner based on matrix multiplication), and this helps the model to scale well such that it can process large graphs efficiently. But the model assumes homogeneous, undirected graphs, and is node-centric, learning only node embeddings, not relation embeddings. Relational GCNs (R-GCNs) (Schlichtkrull et al., 2018) extend standard GCNs to support multi-relational graphs but remain node-centric.

Graph Attention Networks (GATs) (Velickovic et al., 2017) include in the node embedding update function learned attention coefficients α_{ij} that encode the attention to be given by node i to neighbour node j. Standard GATs support homogeneous graphs only (and hence single-type relations only) and are node-centric. Relational GATs (R-GATs)

(*e.g.*, M. Chen et al., 2021) extend standard GATs to support multi-relational graphs, but remain node-centric. The node embedding update function uses both attention coefficients and learned, relation-specific transformation matrices.

CompGCN (Vashishth et al., 2020) extends GCNs in order to more fully support multi-relational graphs by jointly embedding both nodes and relations. Relation embeddings are learned using a dedicated relation embedding update function based on learned, relation-specific transformation matrices. Entity-relation composition operations are employed in the node embedding update function whereby the embeddings for edges in the neighbourhood of a node are composed with the embedding for that node. The composition is direction-aware and provides support for inverse relations.

Recall from our earlier discussion of *knowledge* graphs (KGs) that one of their defining characteristics is that they are multi-relational: *i.e.*, relations between entities can be of different type and carry different semantics. Recall also that we described several knowledge graph embedding (KGE) models designed to learn embeddings for (multi-relational) knowledge graphs. Given that GNNs and KGE models share the same goal of learning embeddings for graphs, one way to regard GNNs (generally) is to view them as *precursors* of KGE models: a solution approach focused on a simpler problem (at least, initially). With the emergence of GNN support for multi-relational graphs in models like CompGCN, the distinction between GNNs and KGE models blurs to the point where advanced GNNs (like CompGCN) can reasonably be regarded as representing a deep learning-based and message passing-based category of KGE model. Indeed, for CompGCN, Vashishth et al. (2020) refer explicitly to adapting the composition operations used by various KGE models such as TransE (described earlier).

The strong relationship between GNNs and KGEMs is further reinforced by the strong overlap in their applications. The key applications of GNNs concern prediction tasks that fall into three categories: node-level tasks, edge-level tasks, and graph-level tasks. Node-level tasks focus on node embeddings and include: *(i)* node classification (where the goal is to predict a label for each node), *(ii)* node regression (predict a real number for each node), and *(iii)* community detection (measuring similarity and grouping nodes into clusters). Edge-level tasks include: *(i)* edge (or link) prediction (where the goal is to infer links between nodes) and *(ii)* edge classification (predicting not just a link but its type, or label, as well). Graph-level tasks include: *(i)* graph classification (predicting a label for a sub-graph or entire graph), *(ii)* graph regression (predicting a continuous property of a sub-graph or entire graph), and *(iii)* graph matching (where the similarity of graphs is measured by comparing their aggregate graph embeddings).

Conceptually, the GNN application closest to our neurosymbolic research is the edge-level task of *edge classification*. In theory at least, it may be feasible for GNN models capable of undertaking multi-relational edge (or link) classification to act as counter-

parts of our Semantic Web (OWL-based knowledge graph) symbolic reasoning engine in thread two of our research, which explores the task of detecting visual relationships in images. Instead of using OWL-based knowledge graph reasoning to logically expand the training image scene graphs used to supervise the (subsymbolic) learning of our predicate detector, one could conceivably explore the efficacy of GNN-based edge classification at expanding the scene graphs. Similarly, instead of using OWL-based knowledge graph reasoning to validate predictions of visual relationships, one could conceivably use a (suitably equipped) GNN to score the acceptability (or reasonableness) of a predicted visual relationship, as a proxy for evaluating its semantic validity according to OWL ontology VRD-World. However, since our primary research interest is focused on exploring the use of OWL for both symbolic knowledge representation and exact, symbolic logical reasoning, the degree to which GNNs might be capable of simulating these symbolic OWL reasoning tasks has yet to be explored.

2.5.4 Gunther Schmidt and relational mathematics

Gunther Schmidt is a mathematician and computer scientist whose fields of specialisation include relational mathematics and building software to leverage and explore relational mathematics. His works on relational mathematics include: *(i)* Schmidt and Ströhlein (1993), which highlights the close relationship between relation theory and graph theory (two sides of the same coin); *(ii)* Schmidt (2011), a comprehensive examination of relational mathematics, from its roots in the *Calculus of Relations* through to modern relation algebras; and *(iii)* Schmidt and Winter (2014), an addendum to the work just cited that records further specialised relational results. In their synopsis of Schmidt's academic career (Berghammer & Winter, 2014), two of Schmidt's students credit him (along with others, such as Steven Givant (Givant, 2017)) of raising the profile of the *Calculus of Relations* in the 20th century, and with helping to develop it into modern relation algebra.

The field of mathematics referred to by the (little known) phrase *the Calculus of Relations* is the foundation of modern relational mathematics and relation algebras. Givant (2017) provides a brief history of the Calculus of Relations in the book's introduction, and devotes the opening chapter to providing a thorough overview. The Calculus of Relations codifies the many operations, laws and properties that hold for binary relations. Core relational operations include: relational intersection, relational union, relational complement, relational converse, and relational composition. The theory of relation algebras is described as being akin to an abstract Calculus of Relations.

In Givant (2017), the origins of the Calculus of Relations are traced from Augustus DeMorgan, to Charles Pierce (who Givant credits as the 'creator of the theory of relations', between 1870 and 1882), to Ernst Schröder (who provided the only exhaustive treatment of the subject), and to its coverage in Whitehead and Russell's 'Principia Mathe-

matica', in 1903, where it is described as one of the three parts upon which symbolic logic is founded. After that, the Calculus of Relations attracted relatively little attention, however, until the 1940s when mathematician and logician Alfred Tarski sought to revitalise the subject with a paper entitled 'On the calculus of relations' (Tarski, 1941). Tarski took an axiomatic approach to the subject and eventually reduced his system to just 10 equational axioms that express laws that hold in all algebras of binary relations. From these, Tarski derives Schröder's hundreds of laws. Givant (a student of Tarski) says that, from 1945 onward, Tarski and he jointly developed the Calculus of Relations into a theory of *relation algebras*. Several works of Gunther Schmidt are cited in Givant (2017) but, in the book's acknowledgements, Schmidt is mentioned only in relation to having provided references to "literature concerning applications of the theory of relation algebras to computer science", not for having contributed to the development of the theory itself. Determining whether and how credit for developing the Calculus of Relations into the modern theory of relation algebras should be shared between Givant and Schmidt requires further independent investigation. We have focused on the Calculus of Relations here because it provides the mathematical underpinning and justification for thread three of our research concerning tensor knowledge graphs and tensor knowledge graph reasoning.

In the Introduction to his book on relational mathematics (Schmidt, 2011), Schmidt says: "Hardly anybody confronted with practical problems knows how to apply relational calculi; there is almost no broadly available computer support." This statement helps to explain the motivations behind the several software systems for relational mathematics developed by Schmidt and his students. One such system, RELVIEW (*e.g.*, Behnke et al. (1998), Berghammer and Neumann (2005), Berghammer and Schmidt (1993, 2007), and Berghammer et al. (1996)), is an interactive, GUI-based system that permits graphical manipulation of relations as Boolean matrices. It also provides support for things such as proving or disproving relational results or properties. Another system, RATH (Kahl & Schmidt, 2000), is a library of Haskell modules that facilitates exploration of relation algebras by providing a variety of means for constructing and testing such algebras. A third system, TituRel (*e.g.*, Schmidt (2003, 2004)), was conceived as a *relational reference language* for working with relations and relational mathematics generally. Originally named the *Relation Language*, the evidence points to TituRel being a personal project of Schmidt's. His ambitions for TituRel were for it to support the formulation of all problems for which relational methods are known to have provided solutions. The language is accompanied by a library of Haskell modules providing specialised relational services to facilitate the construction of task-specific software systems based on relational mathematics.

Our concepts of tensor knowledge graph (TKG), tensor knowledge graph reasoning (TKGR), and tensor knowledge graph reasoning engine (TKGRE) are similarly rooted

in relational mathematics, but they are quite different in nature to Schmidt's systems, and language, just described. A TKG is a relational data structure, but one that represents the knowledge of an arbitrary OWL-based knowledge graph. TKGR is based on basic binary relational operations, as formalised by the Calculus of Relations, but it performs and emulates OWL (*i.e.*, $SROIQ$ Description logic) reasoning. A TKGRE can be said to be a relational software system, but one that hosts a TKG, and applies TKGR, and for the purpose of providing useful symbolic OWL reasoning services in neurosymbolic systems (whether in batch mode or real-time). It may be that Schmidt's relational language TituRel is capable of implementing Haskell counterparts of all three of our concepts. But we cannot say since TituRel is not publicly available. In comments attached to a post on MathOverflow asking "What happened to TituRel?"[1], in 2018, one user relays a response from having emailed the question directly to Schmidt himself. Schmidt is said to have replied: "TituRel is not available as downloadable software due to not much interest from the outside world."

In the Preface to his book on relational mathematics (Schmidt, 2011), Schmidt remarks that since relations are used so widely, whether to express, or model, or reason, or compute with, "it sometimes looks as if the wheel is being reinvented when standard results are rediscovered in a new specialized context". We believe this to be the correct way to view our contributions regarding tensor knowledge graphs and tensor knowledge graph reasoning (the subjects of thread three of our research). We have rediscovered operations and laws known to relational mathematics (and to its foundational theory, the Calculus of Relations) and successfully applied and adapted these in the process of exploring novel approaches to representing OWL-based knowledge graphs and to performing OWL (aka Description logic) reasoning. Hence, while our contributions in this regard are not theoretical, they are novel applications of independently rediscovered (known) relational theory.

[1] https://mathoverflow.net/questions/313206/what-happened-to-titurel

Chapter 3

NeSy4VRD

Recall that our research exploring neurosymbolic learning and reasoning with OWL-based knowledge graphs has three threads. This chapter presents the first of these threads: our NeSy4VRD resource. The acronym *NeSy4VRD* stands for *Neurosymbolic AI for Visual Relationship Detection*.

NeSy4VRD is an 'image dataset + OWL ontology' resource that enables neurosymbolic research with OWL-based knowledge graphs in the computer vision task of detecting visual relationships in images. It is based on the VRD image dataset (Lu et al., 2016) and represents both a significant refinement and significant extension of that dataset. The components of NeSy4VRD evolved organically in the course of enabling ourselves to pursue thread two of our research (described in Chapter 4). We packaged its components into an integrated, re-usable, and documented resource and contributed NeSy4VRD to the computer vision, neurosymbolic AI and Semantic Web communities to enable more neurosymbolic research using OWL-based knowledge graphs. Our contribution of NeSy4VRD was announced in Herron, Jiménez-Ruiz, Tarroni, and Weyde (2023).

We begin this chapter by providing an overview and background for NeSy4VRD. Next, we discuss the VRD image dataset (Lu et al., 2016) upon which NeSy4VRD is based, and we define the concept of *visual relationship* as it pertains to the VRD dataset and to NeSy4VRD, and throughout this document. Then we describe the extensive visual relationship annotation analysis and customisation exercise that we undertook to clean and improve the original crowd-sourced annotated visual relationships of the VRD images. This annotation customisation exercise permitted us to design a precise and credible OWL ontology that faithfully describes the domain of our quality-improved NeSy4VRD annotated visual relationships. This domain is a broad and sparse subset of the real world, as reflected in the common, everyday images of the VRD dataset. In reference to this fact, we named our custom OWL ontology *VRD-World*. We present VRD-World

and describe its characteristics. We conclude with a description of ancillary components of NeSy4VRD, such as its extensive software infrastructure support for the analysis and extensibility of the NeSy4VRD annotated visual relationships and, by extension, of the VRD-World OWL ontology as well.

3.1 Overview

To pursue our vision for thread two of our research, presented in Chapter 4), we had two requirements: *(i)* we wanted an image dataset with a well-aligned companion OWL ontology, and *(ii)* we wanted the alignment between the ontology and the dataset to be strong enough for the results of OWL reasoning to be pertinent to the subsymbolic learning task at hand. Such resources are scarce. So we created one, and called it NeSy4VRD. The scarcity of such resources is surely a barrier to entry for neurosymbolic research using OWL-based knowledge graphs generally, so we contributed NeSy4VRD to the neurosymbolic AI community after having created it.

NeSy4VRD couples the images of the VRD dataset with an extensively revised and quality-improved version of the original crowd-sourced VRD annotated visual relationships. NeSy4VRD accompanies the VRD images and the NeSy4VRD annotated visual relationships with a custom OWL ontology that describes the domain reflected in the NeSy4VRD annotated visual relationships, and thereby in the VRD images as well. We call this custom OWL ontology *VRD-World*. Our OWL ontology VRD-World enables an OWL-based knowledge graph tool to perform symbolic (OWL) reasoning that is directly pertinent to the domain of the NeSy4VRD dataset, and to the task of detecting visual relationships in the VRD images of the NeSy4VRD dataset.

Public access to the original crowd-sourced VRD annotated visual relationships is still available today. But public access to the VRD images themselves disappeared sometime in late 2021 or early 2022 due to computer server restructuring at Stanford University, where the VRD dataset originated. NeSy4VRD restores public access to the VRD images, with permission from one of the principals behind the original VRD dataset, Dr. Ranjay Krishna.

NeSy4VRD provides comprehensive open source software infrastructure that supports the analysis and extensibility of the NeSy4VRD annotated visual relationships. The annotation extensibility support facilitates the extensibility of the companion OWL ontology, VRD-World. Both of these components, the NeSy4VRD annotated visual relationships and the companion OWL ontology, VRD-World, can be used *as is* by neurosymbolic AI researchers if they wish. However, the intention behind NeSy4VRD's analysis and extensibility support is to encourage neurosymbolic systems researchers to consider regarding the annotations, and even the ontology, VRD-World, as *starting*

points to be tailored and tuned to suit particular research needs or interests.

NeSy4VRD is composed of the following components:

(i) the *NeSy4VRD dataset* consists of the VRD images from the VRD dataset together with our quality-improved NeSy4VRD annotated visual relationships;

(ii) our well-aligned, companion OWL ontology, VRD-World, that describes the domain of the NeSy4VRD dataset;

(iii) software infrastructure that supports the extensibility of the NeSy4VRD annotated visual relationships and, by extension, of the VRD-World OWL ontology;

(iv) sample Python code for working with the VRD-World ontology and the NeSy4VRD visual relationships in relation to a Python-based knowledge graph tool;

(v) a description of a vision and process that we call *Distributed Annotation Enhancement* (DAE), for continuously extending the NeSy4VRD annotated visual relationships in a decentralised fashion, over time, through the independent actions of researchers pursuing their independent research interests.

The NeSy4VRD dataset and the VRD-World ontology underpin all of thread two of our research (per Chapter 4), in which we examine using OWL-based knowledge graph reasoning for the computer vision task of visual relationship detection in images (aka image scene graph generation). And our OWL ontology, VRD-World, along with a subset we call mini VRD-World, played important roles in thread three (described in Chapter 5), by helping to drive the research and development of our notions of tensor knowledge graphs and tensor knowledge graph reasoning.

NeSy4VRD is available online. The NeSy4VRD dataset and its companion VRD-World OWL ontology are freely available in the Zenodo repository at https://doi.org/10.5281/zenodo.7916355 under open source license *CC BY 4.0*. The NeSy4VRD software infrastructure that supports the analysis and extensibility of NeSy4VRD itself, along with comprehensive documentation of how it use it, plus comprehensive documentation of our DAE vision for continuous, decentralised enhancement of NeSy4VRD by independent researchers working independently, are freely available on the GitHub platform at https://github.com/djherron/NeSy4VRD under open source license *MIT/Expat*.

3.2 The VRD dataset and visual relationships

As mentioned above, NeSy4VRD is based on the Visual Relationship Detection (VRD) image dataset (Lu et al., 2016). The visual relationships annotated for the VRD images are 5-tuples with the following conceptual structure

```
(subj_bbox, subj_class, predicate, obj_bbox, obj_class),
```

where the bounding boxes of the two objects are each represented by four integers, and the two object classes and the predicate are represented by integer labels (indices into ordered lists of object class names and predicate names). The predicate expresses a particular relationship between a particular ordered pair of objects in a particular image.

A raw annotated NeSy4VRD (and VRD) visual relationship, stored in JSON format, looks like this:

```
{'predicate': 9,
 'subject': {'category': 35, 'bbox': [12, 243, 659, 724]},
 'object': {'category': 23, 'bbox': [253, 409, 426, 675]}}.
```

Visual relationship instances versus types Throughout this document we frequently refer to example visual relationships in order to discuss concepts and illustrate aspects of OWL reasoning. Whenever we do so, for simplicity we adopt a more compact and user-friendly 3-tuple representation. It is frequently convenient to distinguish between visual relationship *instances* and visual relationship *types*. VR instances are concrete: *i.e.*, NeSy4VRD annotated VRs, VRs inferred by OWL reasoning from NeSy4VRD annotated VRs, and individual predicted VRs that emerge during neural network training or inference. VR types are conceptual: abstractions (generalisations) of VR instances that we entertain, conceptually, when convenient. Every VR instance has a unique corresponding VR type. A VR instance relates two particular objects, using a particular predicate. Its corresponding VR type relates two (corresponding) particular object classes, using the same predicate. We use a user-friendly 3-tuple representation for both VR instances and VR types. In fact, for VR types, only a 3-tuple representation is meaningful. For VR instances we use lower-case identifiers for the subject and the object. For example, suppose image X has the annotated VR (instance) `(personX ride horseX)` and that image Y has a similar (but different) annotated VR (instance) `(personY ride horseY)`. For VR types we use capitalised identifiers for the subject class and object class. Conceptually, these two example VR instances belong to the same VR type: `(Person ride Horse)`, where `Person` and `Horse` are references to NeSy4VRD object classes (and to corresponding classes in OWL ontology VRD-World). Figure 3.1 shows two representative VRD images with their NeSy4VRD annotated objects and some of their NeSy4VRD annotated visual relationships (VR instances, as opposed to VR types).

Scene graphs The collection of VRD (and now NeSy4VRD) visual relationships annotated for an image of the VRD dataset forms a natural *directed graph* of the scene in the image. Perhaps as a result, and as is evident in the survey paper Zhu et al. (2022),

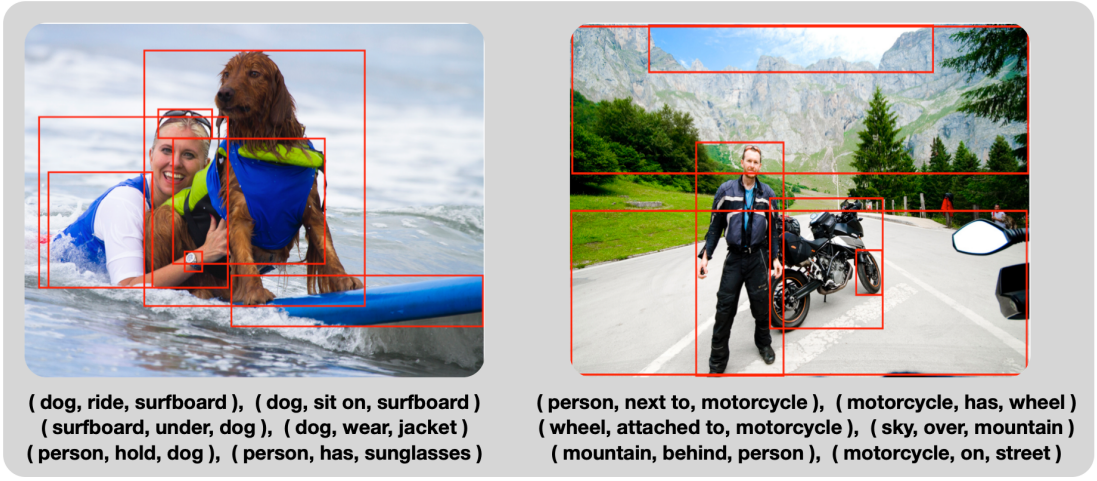


Figure 3.1: Two representative images from the VRD dataset with objects localised in bounding boxes and samples of representative annotated visual relationships rendered as user-friendly 3-tuples.

much of the image interpretation literature speaks of *scene graphs* and *scene graph generation* rather than *visual relationships* and *visual relationship detection*. These language differences in the literature mask the fact that two research communities are essentially talking about the same thing in different ways. We tend to use the language of *visual relationships* because we were exposed to it first, via the VRD dataset and Lu et al. (2016). But we switch to speaking of *scene graphs* at times, when the context suggests it may be helpful.

3.2.1 Attractive characteristics of the VRD dataset

The VRD dataset is not the only image dataset with annotated visual relationships. Table 3.1 summarises the most prominent image datasets whose accompanying annotations include visual relationships (or scene graphs) in some form. Zhu et al. (2022) mention several further, less prominent such datasets, most of which are subsets of Visual Genome.

The VRD image dataset has several characteristics that make it attractive, including for neurosymbolic research. The first is its small size: 4,000 training images and 1,000 test images. Since deep learning is known to be data hungry, the small amount of training data (in theory) leaves greater space for symbolic components (like OWL-based knowledge graphs) to enhance deep learning's predictive performance. Second, relative to its size in images, the number of distinct object classes and predicates referenced in its visual relationships is relatively large: 100 and 70, respectively. In our case, this suggests that the annotations have the potential of leading to the design of a reasonably

Table 3.1: Image Datasets with Annotated Visual Relationships

Dataset	Images	Classes	Predicates	VR types	VR instances
VRD[1]	5,000	100	70	6,672	37,993
Visual Genome[2]	108,077	33,877	42,374	n/a	2,347,000
Open Images v4[3]	9.2m	600	n/a	329	391,000
COCO-a[4]	330,000	81	140	n/a	n/a[5]

[1] Lu et al. (2016)
[2] Krishna et al. (2016)
[3] Kuznetsova et al. (2020). In version 4 (2020), the 600 object classes are annotated within 1.9m of the available images, and the 391,000 annotated visual relationships involved just 57 of the 600 object classes.
[4] Ronchi and Perona (2015). Human agents are the subject of every visual action.
[5] Instances of *n/a* indicate where we were unable to find a count.

complex yet moderately sized ontology: something that requires substantial logical inference to be exercised when OWL reasoning is activated, while being manageable in scope. Third, the distribution of the *types* of the annotated visual relationships,

$$(s_i\ p_k\ o_j) \quad i, j = 1, \ldots, 100 \quad k = 1, \ldots, 70,$$

has a long tail that provides many instances of data conditions suited to the study of zero-shot and few-shot learning. Zero-shot and few-shot learning tasks provide good opportunities for neurosymbolic researchers to explore ways by which symbolic components (like OWL-based knowledge graphs) might improve deep learning's ability to generalise from limited training examples.

The scene graph (and visual relationship detection) survey paper Zhu et al. (2022) shows that many researchers have been attracted to the VRD dataset. Visual Genome is the most popular. The datasets in Table 3.1 share a common shortcoming with respect to our research needs, however: none of them has a well-aligned, companion OWL ontology closely describing the domain of the dataset. We opted to work with the VRD dataset for the reasons noted above, and we resolved to engineer our own companion OWL ontology.

3.2.2 Unattractive characteristics of the VRD dataset

In the course of familiarising ourselves with the VRD images and their annotated visual relationships, we developed extensive software functionality to search for and display images and visual relationships that satisfy myriad data conditions. A key part of this functionality and analysis involved displaying an individual annotated visual relationship against its associated image. We did this by drawing the bounding boxes annotated

for the ordered pair of objects participating in a visual relationship: one colour for the object playing the 'subject' role, and a different colour for the object playing the 'object' role, so visual interpretation was easy. We found this to be the only sure way to evaluate any given annotated visual relationship, in order to:

- verify that bounding boxes are reasonable;
- understand the true range of real-world objects sharing a given object class name;
- understand the true range of senses in which annotators use a given predicate.

During our analysis, we attended to these considerations closely because our objective of designing a precise and credible companion OWL ontology was always uppermost in our mind. Our prime concern was to ensure that we fully understood the semantics of the 100 object class names, and the 70 predicate names, so that we could make informed decisions *(i)* when constructing OWL class hierarchies, *(ii)* when specifying object property characteristics (like symmetry and transitivity) and object property relationships (like inverses, property equivalence, and sub-property hierarchies), and *(iii)* when specifying classes that restrict the domains and/or ranges of object properties in some way.

Our analysis revealed many issues with the crowd-sourced VRD annotated visual relationships, some of which were particularly problematic with respect to our requirements (described above). The more we looked, the more issues accumulated. The main issue categories we discovered within the original VRD annotated visual relationships are these:

(1) stark variability in the types of objects sharing certain object class names;

(2) different object class names for objects clearly drawn from the same distribution and otherwise indistinguishable from one another;

(3) diverse semantics (usage) of single predicate names;

(4) outright errors in visual relationship construction;

(5) multiple near duplicate bounding boxes for the same object in a single image;

(6) multiple near and/or exact duplicate visual relationships annotated for an image;

(7) poor quality bounding boxes.

Category (1) issues make it problematic to design a credible class hierarchy for an OWL ontology. For example:

- object class `bear` pertains to both real bears and teddy bears;

- object class `plate` pertains to dishware plates, vehicle license plates, baseball 'home' plates and bases, plaques on walls, etc.;
- object class `glasses` pertains to eyeglasses, drinking glasses, miscellaneous things made of glass (e.g. table tops, glass barriers, etc.);
- object class `person` pertains to both people and to stereo (audio) speakers.

Category (2) issues make object detection harder and more error prone. For example:

- object classes `plane` and `airplane` label objects that are indistinguishable and are clearly drawn from the same distribution;
- object classes `coat` and `jacket` do the same;
- object classes `road` and `street` do the same.

Category (3) issues make it problematic to define object properties (the ontology counterparts of predicates) precisely, including their particular characteristics, relationships, domains, and ranges. For example:

- predicate `across` is used in the sense of 'along side of', 'is crossing the', 'is across from', etc.;
- predicate `fly` is used in the sense of 'is flying a', 'is flown by', 'is flying in', 'is flying above', etc.;
- predicate `walk` is used in the sense of 'walk on', 'walk next to', 'walk towards each other', 'walking the dog', etc..

Category (4) issues complicate visual relationship detection overall. For example:

- in possessive relationship patterns such as `(person wear X)` or `(person hold X)`, frequently the bounding box for `X` places object `X` on or with a person different from the one referenced in the subject of the relationship;
- in positional relationship patterns such as `(X behind Y)` or `(X above Y)`, frequently `X` and `Y` need to be swapped or the predicate changed to its inverse.

Issue categories (5), (6) and (7) are self-explanatory. Categories (1) and (3) have serious implications with respect to ontology design. If left unresolved, these would make designing a credible ontology infeasible.

Table 3.2: Quantitative comparison of the NeSy4VRD and VRD annotated visual relationships (VRs).

Metric	**NeSy4VRD**	**VRD**
Object classes	109	100
Predicates	71	70
Number of training set annotated VRs	29,333	30,355
Number of test set annotated VRs	9,201	7,638
Total annotated VRs	38,534	37,993
Average VR annotations per training image	7.8	7.6
Average VR annotations per test image	9.9	7.6
Number of training images with duplicate VRs	0	323
Number of test images with duplicate VRs	0	91

3.3 NeSy4VRD annotated visual relationships

NeSy4VRD resolves the problematic issues with the VRD annotated visual relationships discussed in the previous section. With the help of NeSy4VRD tooling, and its semi-automated, repeatable workflow (described shortly), we customised (changed, removed, contributed) visual relationship annotations for 1,715 of the 4,000 VRD training images, and 828 of the 1000 VRD test images, for a total of 2,543 images, or just over half of the total number. Table 3.2 compares the NeSy4VRD annotated visual relationships with the original VRD annotated visual relationships in relation to various quantitative metrics. The table indicates that NeSy4VRD has more object classes, more predicates, more visual relationships overall, and greater average annotated visual relationships per image. The associated quality improvements of the NeSy4VRD annotated visual relationships are harder to convey.

3.4 Our OWL ontology design goals

We engineered our VRD-World OWL ontology with several design goals in mind, each of which has been satisfied. These design goals were:

- define classes for the ontology that are one-to-one counterparts of the NeSy4VRD object classes, and position these as *leaf* classes in a class hierarchy;
- ensure the class hierarchy possesses credible, common sense, and reasonably rich subsumption paths that will exercise the 'type inference' capabilities of OWL reasoning in a meaningful way;
- define object properties for the ontology that are one-to-one counterparts of the

NeSy4VRD visual relationship predicates;

- ensure the object properties have credible, common sense, and reasonably rich characteristics (like symmetry and transitivity), relationships (like inverses, equivalent properties, and sub-property/parent-property relationships), and domain and range restrictions (where applicable), in order to ensure that the 'link inference' capabilities of OWL reasoning will be exercised in a meaningful way;
- ensure that the ontology, overall, contains sufficient supplementary ontological infrastructure (*e.g.*, supporting classes, object properties, and data properties, as required) in order for a knowledge graph, governed by the ontology, to faithfully host the NeSy4VRD annotated visual relationships in their entirety, without loss of information, and which permits them to be extracted from the knowledge graph and fully reconstituted, even if they have been augmented (materialised) by OWL reasoning.

3.5 Overview of the VRD-World OWL ontology

Here we begin describing the OWL ontology that we engineered to be a well-aligned companion to the NeSy4VRD dataset so that we might study the effects that symbolic OWL reasoning can have on neural, subsymbolic learning within the application task of visual relationship detection in images. We called this ontology *VRD-World* because the domain it describes (that of the NeSy4VRD dataset) is a broad and sparse subset of our common, everyday real world.

Table 3.3 provides a quantitative view of the VRD-World OWL ontology (version 1) in terms of key OWL ontology metrics. We can see that while NeSy4VRD has 109 object classes, VRD-World has 239 classes. The extra classes in VRD-World arise because we mapped the 109 NeSy4VRD object classes to counterpart VRD-World classes that appear as *leaf* classes within a larger class hierarchy that we designed around them. The count of SubClassOf axioms is a related metric that gives the number of explicit (asserted) axioms in the ontology that declare one class to be a subclass of another, such as

```
vrd:Car rdfs:subClassOf vrd:RoadMotorisedVehicle .
vrd:Chair rdfs:subClassOf vrd:SeatingFurniture .
vrd:SeatingFurniture rdfs:subClassOf vrd:Furniture .
```

The 71 NeSy4VRD predicates map to counterpart VRD-World object properties, and these form the bulk of the 74 object properties reported in Table 3.3. The 3 extra object properties relate to how we represent images within VRD-World, such that an image and all of its visual relationships (*i.e.*, its scene graph), whether asserted or inferred, can be

Table 3.3: Summary metrics for the VRD-World OWL ontology (v1)

Summary Metric	Count
Axiom	815
Logical axioms	433
Declaration axioms	322
Classes	239
Object properties	74
Data properties	4
Annotation properties	8

Class axioms	Count
SubClassOf	242
EquivalentClasses	21
Object property axioms	
SubObjectPropertyOf	46
EquivalentObjectProperties	7
InverseObjectProperties	6
TransitiveObjectProperties	17
SymmetricObjectProperties	8

stored and managed as a cohesive unit, like a sub-graph of the knowledge graph.

OWL object property axioms are used to declare that a given object property has a certain characteristic, such as symmetry or transitivity, or is involved in certain relationships with other properties, such as having an inverse or a parent property. Two object properties having different names may also be declared to be equivalent to one another. In Table 3.3 we see that, in VRD-World, there are 46 explicit (asserted) axioms declaring sub-property relationships such as:

```
vrd:wear rdfs:subPropertyOf vrd:has .
vrd:sitUnder rdfs:subPropertyOf vrd:under .
```

And there are 7 explicit declarations of pairs of equivalent properties, such as:

```
vrd:beside owl:equivalentProperty vrd:nextTo .
vrd:beneath owl:equivalentProperty vrd:under .
```

For some of the metrics in Table 3.3, the counts represent only what has been explicitly declared (asserted to be true) in the VRD-World ontology, not what is logically entailed and may eventually be materialised during OWL reasoning. OWL reasoning will not infer new classes for the class hierarchy, or new object properties for relating individuals, but it will infer whatever class axioms and object property axioms are entailed by the ontology. The richness of VRD-World in terms of classes, object properties, and object property characteristics and relationships, gives OWL reasoning plenty of scope to perform substantial amounts of logical inference when activated.

3.6 The VRD-World class hierarchy

Figure 3.2 provides a view of the class hierarchy of our VRD-world OWL ontology (version 1). This is a custom class hierarchy that we designed around the NeSy4VRD object

classes. The NeSy4VRD object classes are *leaf* classes in the hierarchy. All of the parent classes in the hierarchy were introduced by us in order to unify the NeSy4VRD object classes into a coherent whole. We took a bottom-up approach to arranging the hierarchy.

Our Wikidata-inspired class hierarchy for VRD-World In addition to the class hierarchy for the VRD-World OWL ontology discussed so far, in the early stages of our research we also designed an alternate class hierarchy for VRD-World, one inspired by Wikidata (Vrandečić & Krötzsch, 2014). We mapped each NeSy4VRD object class to a counterpart class in the vast Wikidata class hierarchy, and then used SPARQL queries to find all the Wikidata parent classes of these 'base' classes. We wrote recursive code to isolate each of the unique subsumption paths implied by the bag of Wikidata child-class/parent-class links returned by our SPARQL queries. We browsed these subsumption paths visually and, from amongst them, selected a few that blended well with the nascent class hierarchy for VRD-World that gradually emerged. We have not used this potential alternate class hierarchy in our research. Appendix A describes this alternate, Wikidata-inspired class hierarchy for VRD-World in more detail.

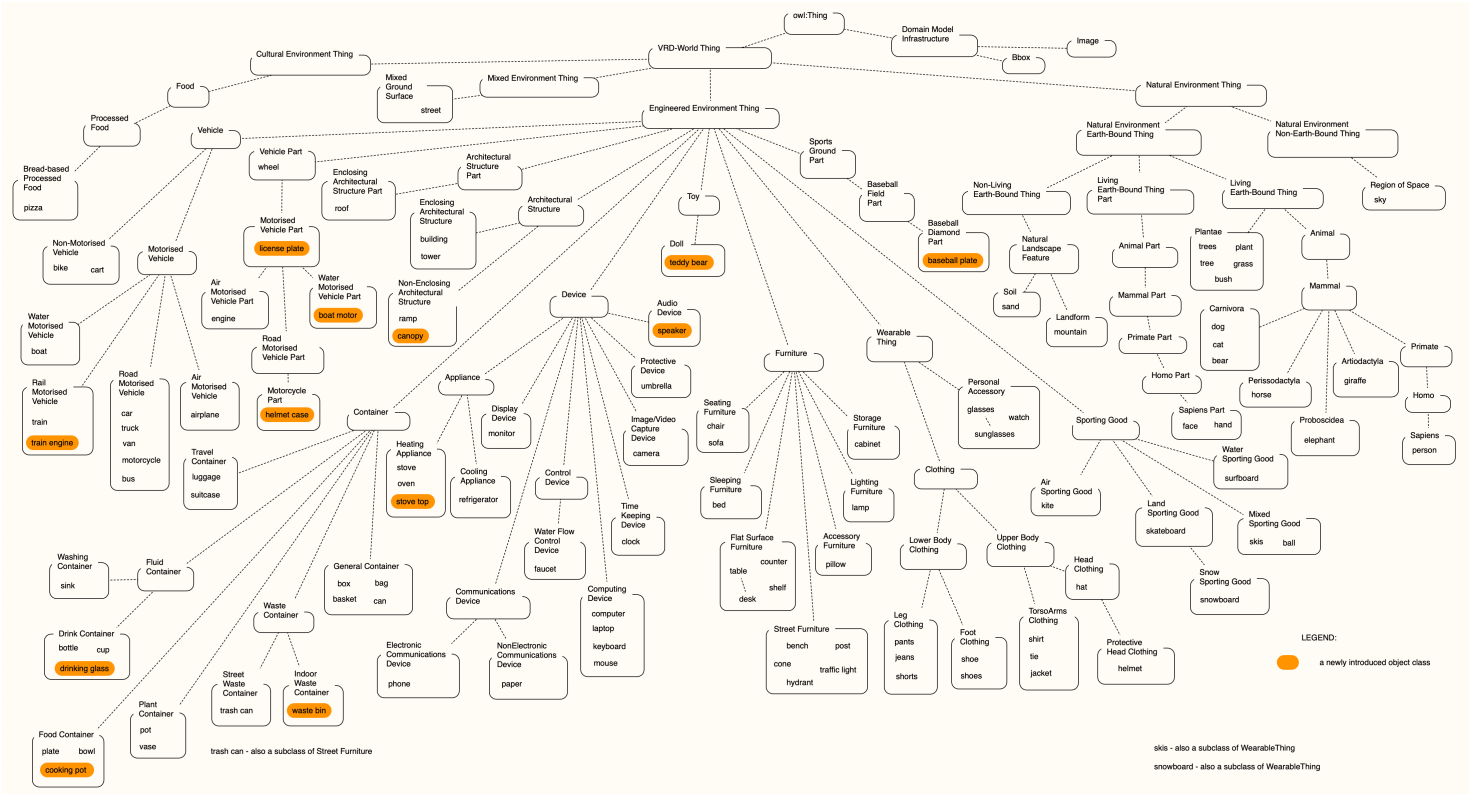


Figure 3.2: A partial pictorial rendering of the class hierarchy of the VRD-world OWL ontology (version 1). The object classes of the NeSy4VRD dataset are represented one-to-one by leaf classes in the hierarchy. To minimise clutter, the names of these leaf class are *enclosed within* the bubble that represents their main parent class. Lines represent sub-class/parent-class relationships. Classes coloured orange are new to NeSy4VRD and do not exist in the annotated visual relationships of the VRD dataset. All bubbles represent parent classes that were introduced to organise the NeSy4VRD leaf classes into a coherent hierarchy. To minimise clutter, this rendering of the VRD-World class hierarchy does *not* reflect *(i)* many subclass relationships, and *(ii)* many additional classes declared to be unions of other classes.

3.7 The VRD-World object properties

Figure 3.3 provides a view of the object properties of our VRD-world OWL ontology (version 1). The VRD-World object properties are one-to-one counterparts of the predicates used in the annotated visual relationships of NeSy4VRD. The object properties (predicates) are either common spatial relations or common verbs.

Unlike class hierarchies for VRD-World, where we engineered two alternate candidates (as described above), we designed only one version of object properties for VRD-World. One reason is that, with object properties, there was no requirement to introduce additional, more general properties in order to unify the base object properties into some all-inclusive hierarchy. And, as can be seen in Figure 3.3, the existing object properties are already fairly rich in common sense subproperty and equivalence relationships.

Another reason is that the common sense semantics of the object properties are relatively fixed, so there is less opportunity for exploring variations whilst maintaining a common sense grounding. For example, in any version of our VRD-World object properties that we might devise, common sense dictates that we would declare property `beside` to be symmetric. Similarly, common sense dictates that we would declare property `sit next to` to be a subproperty of `next to`. That said, many of our choices for defining the object properties of VRD-World are highly subjective and arbitrary. For example, we declare property `over` to be a subproperty of `above`. But it could be the other way round; or they could be declared to be equivalent; or we could pretend they are not related. We explore the malleable nature of the object properties declared for VRD-World in thread two of our research, in Chapter 4, when we examine the effects of OWL reasoning on the task of detecting visual relationships in images.

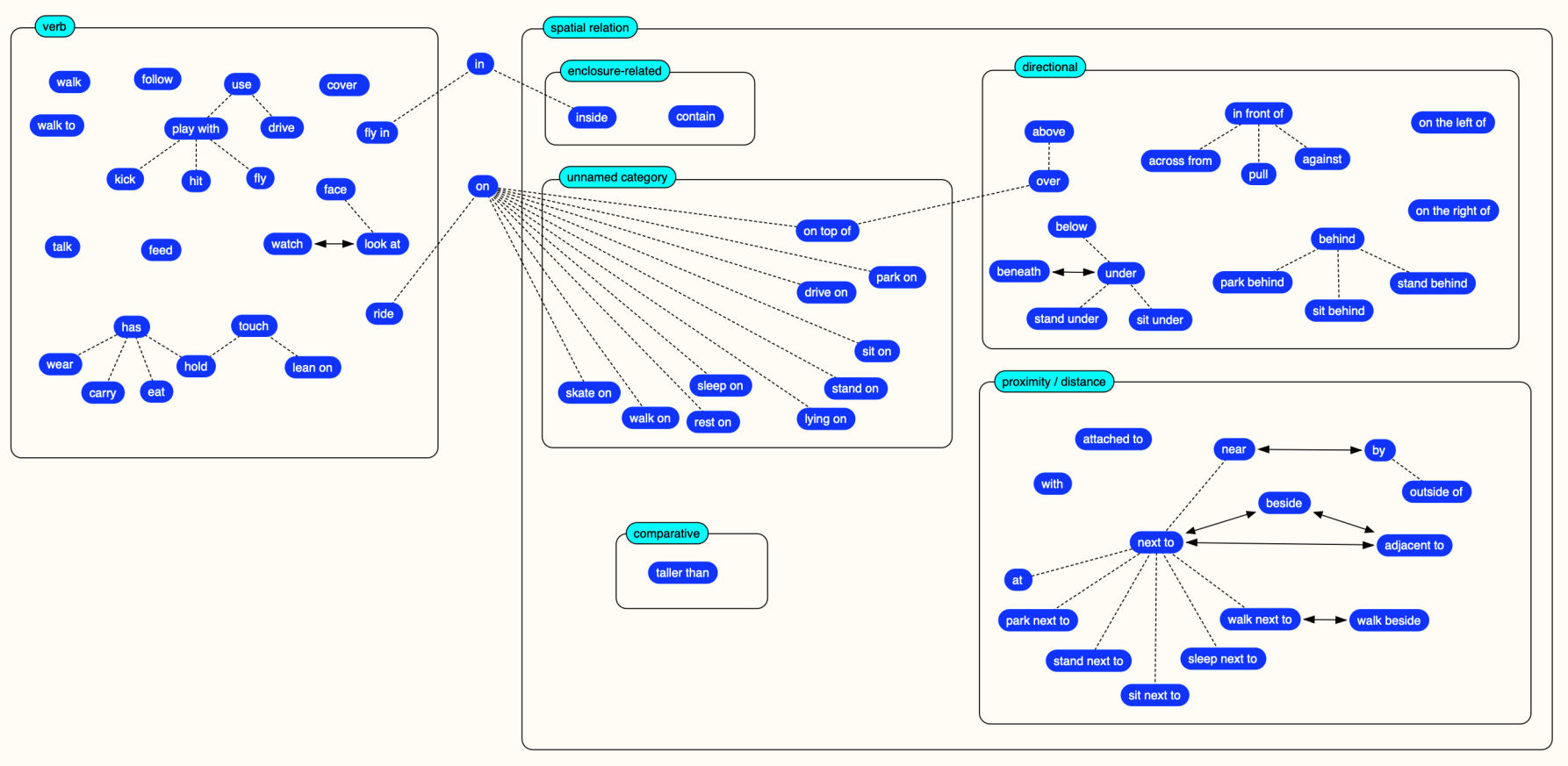


Figure 3.3: A partial pictorial rendering of the object properties of the VRD-world OWL ontology (version 1). Each NeSy4VRD visual relationship *predicate* is represented by a VRD-World object property coloured dark blue. Object properties that share semantics are gathered into bubbles that reflect their semantic category, coloured turquoise. Dotted lines represent sub-property relationships. Solid lines with double arrows represent semantic equivalence. To minimise clutter, this rendering does *not* attempt to express object property relationships such as inverses, or property characteristics such as symmetry and transitivity.

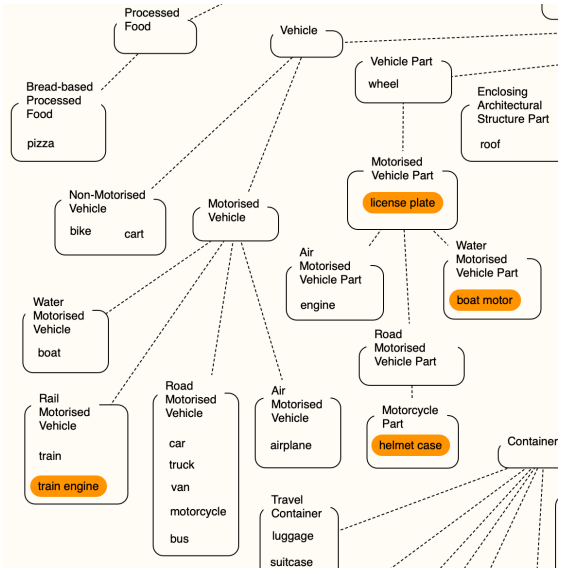


Figure 3.4: A zoom-in on the top-left portion of the VRD-world ontology class hierarchy shown in Figure 3.2. This zoom-in highlights classes `Vehicle` and `VehiclePart`, and the branches of subclasses beneath them.

3.8 Interpreting VRD-World

To help readers get a sense of how the class hierarchy and the object properties of the VRD-World OWL ontology intertwine and interoperate, we briefly discuss how VRD-World models part-whole relations. Classes of 'whole' objects (like `Car`) and classes of 'part' objects (like `Wheel`) are modelled in independent but related branches of the class hierarchy. For example, Figure 3.4 zooms-in on the top-left portion of the full class hierarchy shown in Figure 3.2 to highlight the classes `Vehicle` and `VehiclePart`. Each of these classes is the root of a sub-tree within the overall hierarchy. Under class `Vehicle` we have subclasses representing 'whole' objects such as `Car`, `Airplane`, and `Boat`. Under class `VehiclePart` we have subclasses representing 'part' objects such as `Wheel`, `Engine` (which represents airplane engines only), and `Boat Motor`.

Part-whole relations are modelled using object properties to relate individual 'whole' objects with individual 'part' objects. The VRD-World object property that most closely and exclusively resembles a 'partOf' relation is object property `attachedTo`. For example, within NeSy4VRD we encounter annotated visual relationships such as

```
(wheel attachedTo car)
(engine attachedTo airplane)
(boatMotor attachedTo boat).
```

VRD-World object property `has` (the counterpart of NeSy4VRD predicate `has`) is lin-

guistically flexible enough to often serve as an inverse 'hasPart' relation, leading to counterpart inverse visual relationships such as

```
(car has wheel)
(airplane has engine)
(boat has boatMotor).
```

But note that the natural language semantics of VRD-World object property `has` are too general for one to assume that it *always* connotes a 'hasPart' relation. For example, `has` can also convey the notion of *possession*, as in `(person, has, phone)`; or the notion of *wearing*, as in `(person, has, tie)`; etc.. Therefore, in VRD-World, the object properties `attachedTo` and `has` are *not* declared to be inverses of one another, even though they can sometimes act as inverses.

3.9 NeSy4VRD support for extensibility

For researchers using NeSy4VRD, the default option is to use the NeSy4VRD annotated visual relationships, and the companion OWL ontology, VRD-World, *as is*. But NeSy4VRD itself invites researchers to do otherwise. It invites them to customise and extend NeSy4VRD, to tailor it for their particular research needs, and to share their extensions. In this section, we describe the components of NeSy4VRD that together enable *(i)* the extensibility of the NeSy4VRD annotated visual relationships and, thereby, of the VRD-World OWL ontology as well, and *(ii)* the sharing of independent NeSy4VRD extensions, and the utilisation of shared extensions by others.

NeSy4VRD's support for extensibility comes in the form of *(i)* comprehensive Python code enabling deep but easy analysis of the images and their annotations, *(ii)* a custom, text-based protocol for specifying visual relationship annotation customisation instructions, declaratively, *(iii)* a configurable, multi-step Python-based workflow for applying specified visual relationship annotation customisations and extensions, in an automated, repeatable process, and *(iv)* a vision for a process of decentralised dataset enrichment we call distributed annotation enhancement (DAE).

3.9.1 Comprehensive code for dataset analysis

A prerequisite for customising the NeSy4VRD annotated visual relationships in a systematic way is being able to thoroughly analyse them, in conjunction with the associated images. NeSy4VRD provides comprehensive Python code for this. We list a small sampling of the basic functionality available in NeSy4VRD in order to give a flavour of the visual relationship analysis services it provides. With NeSy4VRD's support for analysing the NeSy4VRD dataset, one can do things such as:

- display an image, showing the objects annotated for: *(i)* one particular visual relationship, *(ii)* a subset of the visual relationships, or *(iii)* all visual relationships;
- print all of the visual relationships for an image: *(i)* in readable `(s p o)` format, *(ii)* in raw format;
- find all images having visual relationships: *(i)* involving object class `X`, or object classes `[X,Y,...]`, *(ii)* using a particular predicate, *(iii)* using a particular visual relationship pattern, such as `(s p X)`, `(X p o)`, `(s X o)`, etc.;
- find all images with a target number of visual relationships;
- perform distributional analyses of: *(i)* the number of visual relationships per image, *(ii)* the number of distinct object classes per image, *(iii)* the number of distinct predicates per image, etc.;
- perform quality verification analyses: *e.g.*, verify that there are no images with *(i)* duplicate visual relationships, *(ii)* degenerate bounding boxes, *(iii)* bounding boxes assigned to multiple different object classes, etc..

3.9.2 The NeSy4VRD protocol

NeSy4VRD provides a custom text-based protocol for specifying changes to, or extensions of, the visual relationships annotated for its images. Specifying customisations and extensions in this way, rather than applying them in an adhoc fashion, facilitates large volumes of customisations or extensions to be applied safely, using the automated, repeatable process of the NeSy4VRD workflow. The NeSy4VRD protocol allows one to specify instructions to *change* existing annotated visual relationships, to *remove* unwanted visual relationships, and to *add* new visual relationships for any image. Listing 3.1 shows representative customisation and extension instructions in relation to five images. These declarative instructions are expressed using the NeSy4VRD protocol.

Listing 3.1: Example visual relationship annotation customisation and extension instructions expressed using the NeSy4VRD protocol.

```
imname; 3223670633_7d3d72dfe8_b.jpg
cvrsoc; 4; (‘person’, ‘on’, ‘shelf’); speaker
cvrsbb; 4; (‘speaker’, ‘on’, ‘shelf’); [161,234,231,270]

imname; 8934043045_251b42d19a_b.jpg
cvrooc; 7; (‘bus’, ‘beside’, ‘car’); truck
cvrobb; 7; (‘bus’, ‘beside’, ‘truck’); [334,557,99,403]

imname; 1426904233_ee344879b6_b.jpg
cvrsoc; 5; (‘bear’, ‘sit on’, ‘basket’); teddy bear
cvrpxx; 5; (‘teddy bear’, ‘sit on’, ‘basket’); in
```

```
imname; 4929276486_ca06aedbb9_b.jpg
rvrxxx; 4; (‘person’, ‘wear’, ‘jacket’);
avrxxx; boat; [477,594,319,746]; has; dog; [478,529,587,618]
avrxxx; boat; [477,594,319,746]; carry; dog; [478,529,587,618]

imname; 7171463996_900cb4ce33_b.jpg; rimxxx
```

In Listing 3.1, the first entry specifies two improvements to a single visual relationship for the image in question (indicated by keyword `imname`). The visual relationship referred to is the one currently occupying position (index) 4 in the image's list of annotated visual relationships. The user-friendly version of this visual relationship is currently `(person on shelf)`. If the NeSy4VRD workflow finds a mismatch between index 4 and `(person on shelf)`, it aborts and explains the discrepancy, allowing corrections to be applied to the instructions in the text file. The `cvrsoc` instruction declares an intention to change the subject's object class, indicated by `soc` in the instruction name, from `person` to `speaker`, (aka 'audio speaker'). Once the NeSy4VRD workflow executes the `cvrsoc` instruction, the user-friendly version of the visual relationship becomes `(speaker on shelf)`, so this is echoed in the instruction which follows, to avoid a mismatch that would lead to the NeSy4VRD workflow aborting. The next instruction for that same visual relationship, the `cvrsbb` instruction, declares an intention to change the subject's bounding box, as indicated by `sbb` in the instruction name.

The other instructions seen in the listing work similarly. Instruction `cvrooc` declares an intention to change the 'object' object class; instruction `cvrobb` declares an intention to change the bounding box of the 'object' object; instruction `cvrpxx` declares an intention to change the predicate of a visual relationship. Instruction `avrxxx` declares an intention to *append* (introduce) a new visual relationship to the image's list of annotations. Instruction `rvrxxx` declares an intention to remove a particular visual relationship. Finally, instruction `rimxxx` requests that an image, and all of its annotated visual relationships, be removed from the annotations altogether.

3.9.3 The NeSy4VRD workflow

The NeSy4VRD workflow is a set of Python modules and scripts that implement a configurable, multi-step sequential process for applying planned, pre-specified customisations and extensions to NeSy4VRD annotated visual relationships, in an automated, safe, repeatable manner. Each sequential step of the workflow relates to discrete category of annotation customisation, and is performed by a dedicated Python script designed for that task. A configuration module co-ordinates the multi-step process. Each step (script) of the workflow imports the configuration module to access the variables it needs in order to execute properly. Each step is optional, and depends only on the

category of change one wishes to apply.

The instance of the NeSy4VRD workflow that we configured for our work—the work that related to cleaning and improving the crowd-sourced annotated visual relationships of the VRD dataset so we could design VRD-World—consisted of the following steps:

1. change existing and/or add new object class names or predicate names to the respective master lists that define these names;
2. execute the annotation customisation and extension instructions that have been specified using the NeSy4VRD protocol, and stored in a particular text file;
3. for a specified set of images, change all instances of object class X to object class Y; (more than one set of images can be processed in this way);
4. merge all instances of object class S into object class R; merge all instances of predicate A into predicate B; (multiple such pairs can be processed);
5. remove all instances of specified visual relationship *types*, globally;
6. remove all image entries from the annotations dictionary that have zero visual relationships annotated for them;
7. execute the annotation customisation and extension instructions that have been specified using the NeSy4VRD protocol, and stored in a particular text file;
8. execute the annotation customisation and extension instructions that have been specified using the NeSy4VRD protocol, and stored in a particular text file;
9. change visual relationship *type* E to *type* F, globally; (multiple such pairs can be processed; restrictive conditions apply);
10. find images with duplicate visual relationships and remove the duplicates, globally;
11. execute the annotation customisation and extension instructions that have been specified using the NeSy4VRD protocol, and stored in a particular text file.

3.9.4 Distributed annotation enhancement

Distributed annotation enhancement (DAE) is the name we have given to (what we believe to be) a novel model for collaboratively enriching the annotations of a dataset, in a decentralised, distributed manner. The idea for DAE emerged from the NeSy4VRD protocol and the NeSy4VRD workflow infrastructure described in the preceding sections. DAE facilitates the sharing of extensions to NeSy4VRD, and the utilisation of shared

extensions. The DAE model is distinct from the conventional crowd-sourcing model for distributing the effort of annotating datasets.

We describe our vision of DAE with respect to NeSy4VRD in two parts. Part one is as follows. Ideally, some researchers opt to use NeSy4VRD in their neurosymbolic systems research. In the process of doing their research, some opt to enrich NeSy4VRD, by extending or tailoring its annotated visual relationships in some way. The motivation may be to tailor NeSy4VRD to better fit particular research needs (*e.g.*, by creating conditions suited for specific experiments), or it may be simply to contribute to the enrichment of the dataset. Ambitions to tailor NeSy4VRD in such ways are realistic because the dataset is small (5000 images), and because NeSy4VRD provides the infrastructure that supports its own extensibility. Ideally, some of the researchers who opt to extend or tailor NeSy4VRD, will go further an opt to share their extensions. The documentation for DAE on GitHub specifies how such extensions can be packaged for sharing. The NeSy4VRD GitHub site also invites researchers to use the site as a central repository for hosting NeSy4VRD extensions or for advertising links to NeSy4VRD extensions.

Part two of DAE is about utilising shared extensions to NeSy4VRD annotations. The vision here is for researchers interested in using NeSy4VRD to have the freedom to *compose* a particular instance of NeSy4VRD annotations that best suits their research needs. The foundation would likely always be the baseline NeSy4VRD annotations, but these would be composable with any or all of the extensions that have been shared up to that point. The NeSy4VRD workflow provides the infrastructure to manage the composition. In particular, it provides the interpreter of the NeSy4VRD protocol. The protocol interpreter permits any NeSy4VRD extension (that is properly specified using the NeSy4VRD protocol) to be composed with the baseline NeSy4VRD annotations, and with any other shared extension of the NeSy4VRD annotations.

For example, suppose researcher A annotates instances of 'water' in the images of NeSy4VRD, and introduces visual relationship annotations that reference water, and opts to share this extension. Suppose researcher B does the same with respect to the object class 'snow'. Researcher C would then have the option to compose an instance of NeSy4VRD that includes the baseline annotations, plus the extensions contributed by researchers A and B.

The concept of DAE is described at greater length in the NeSy4VRD repository on GitHub, at https://github.com/djherron/NeSy4VRD. The concept generalises beyond NeSy4VRD itself. The keys to the idea are the text-based protocol for specifying annotation customisations and extensions, and the protocol interpreter.

3.10 Summary

NeSy4VRD is a multi-faceted 'image dataset + OWL ontology' resource. It facilitates neurosymbolic research with OWL-based knowledge graphs for the computer vision task of detecting visual relationships in images. The NeSy4VRD dataset restores public access to the images of the VRD dataset (Lu et al., 2016), and combines these with quality-improved NeSy4VRD annotated visual relationships. Uniquely, NeSy4VRD accompanies its dataset with a well-aligned, companion OWL ontology, called VRD-World. The alignment is deliberately strong enough for the results of OWL reasoning over the ontology (in the presence of accompanying data facts) to be directly pertinent to the subsymbolic learning task of detecting visual relationships in images. These core components of NeSy4VRD are themselves accompanied by comprehensive software infrastructure that supports the extensibility of NeSy4VRD. NeSy4VRD enables neurosymbolic research with OWL-based knowledge graphs by lowering the barriers to entry for conducting such research.

Chapter 4

Knowledge Graph Reasoning for Visual Relationship Detection

Chapter 3 described thread one of our research into combining subsymbolic learning with symbolic OWL reasoning: our creation of the NeSy4VRD resource to enable our research. This chapter presents thread two of our research: using OWL-based knowledge graph reasoning to guide neural, subsymbolic learning for detecting visual relationships in images. Thread two of our research relies upon NeSy4VRD. Thread two of our research is our first opportunity to address our central research questions: *(i)* how can we combine subsymbolic learning with symbolic OWL reasoning, and *(ii)* what are the effects or benefits of doing so?

To consider our research questions within the setting of thread two (detection of visual relationships in images), we conducted two investigations. Both investigations use a common, baseline deep learning system for visual relationship detection, and specialise that baseline system by *(i)* exercising a particular aspect of OWL reasoning, and *(ii)* by integrating that aspect of OWL reasoning in a particular way. In the process, the baseline deep learning system becomes a neurosymbolic system. Both investigations look for evidence of the effects of OWL reasoning in recall@N predictive performance scores, and by analysing the underlying predicted visual relationships themselves. We look for the effects of OWL reasoning on deep (subsymbolic) learning using other metrics as well, such as: the volume of predicted VRs per image, mAP@N, learning speed (measured in training epochs), and the incidence rate of semantically invalid predicted VRs. Both investigations leverage OWL reasoning during neurosymbolic system training and inference.

For both investigation 1 and investigation 2, the only difference between the common baseline deep learning system and the neurosymbolic system used in the investigation

is the category of OWL reasoning that is exercised, and the manner in which it is integrated. This deliberate construction allows us to be confident that any changes we observe in neurosymbolic system predictive performance relative to baseline, as a result of integrating OWL reasoning, are due to OWL reasoning alone. In other words, we are confident that the effects of OWL reasoning that we observe are *causal*, and not distorted by confounding variables.

We begin the chapter by quickly defining the computer vision task of detecting visual relationships in images, in order to set the scene for all of thread two of our research. Then we describe our baseline deep learning system for detecting visual relationships in images. Then we present investigation 1. Investigation 1 leverages OWL reasoning to augment the visual relationship annotations of the images that serve as targets during training. We show that this augmentation (this scene graph materialisation) enriches the supervision in ways that usually lead to improved visual relationship detection predictive performance, but not always.

Then we present investigation 2. Investigation 2 leverages OWL reasoning to apply a semantic loss penalty as a form of logical constraint, to help teach a neural network the domain and range restrictions declared within the VRD-World ontology in order to avoid predicting visual relationships that are semantically invalid. We show that an OWL-based knowledge graph can be leveraged in the guise of a symbolic reasoning binary classifier. We describe our use of the symbolic reasoning binary classifier during both neural network training and inference. During training, it allows us to apply our semantic loss penalty to guide neural learning. During inference, it allows us to filter-out semantically invalid predictions so they are not submitted for performance evaluation. Following our discussion of investigation 2, we briefly describe some future work, before closing with a summary.

4.1 The visual relationship detection problem

Figure 4.1 sets the scene for all of thread two of our research into combining subsymbolic learning with symbolic OWL reasoning. It provides a conceptual overview of the visual relationship detection problem, where the objective is to identify the objects in an image, and to predict relationships between those objects, such that the predicted visual relationships match the visual relationships that have been annotated for the image. It also expresses the central hypothesis of thread two of our research: that a neurosymbolic system that leverages OWL-based knowledge graphs and OWL reasoning, can outperform a deep learning system that relies on subsymbolic learning alone.

For example, Figure 4.1 indicates that `(dog ride surfboard)` is a visual relationship that has been annotated for the image in the figure. It suggests that the deep learning

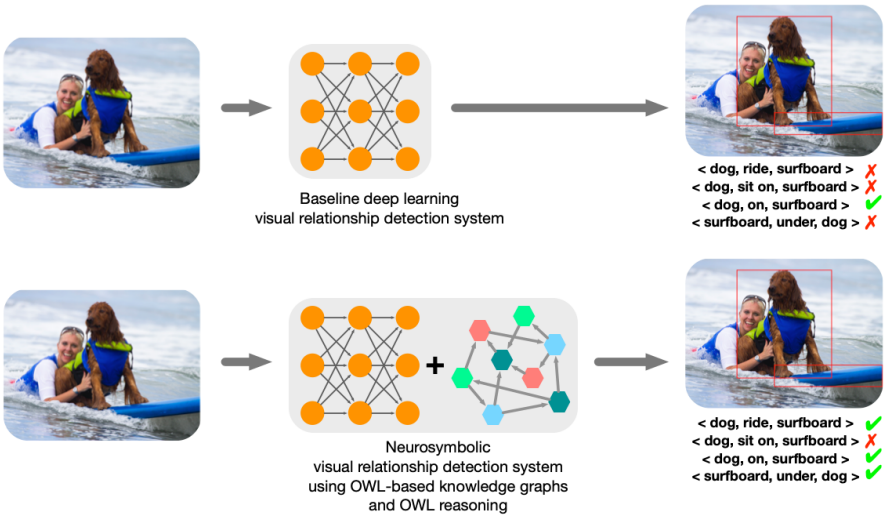


Figure 4.1: A conceptual rendering of the visual relationship detection problem, and of the hypothesis that by combining symbolic OWL reasoning with deep learning, we can improve deep learning's predictive performance with respect to visual relationship detection in images.

system (in the top half of the figure) has failed to predict that visual relationship, and that the neurosymbolic system (in the bottom half), that leverages OWL reasoning in order to guide deep learning, succeeds in predicting that visual relationship.

Metrics traditionally used with the visual relationship detection problem (and the VRD dataset) are recall@50 and recall@100. In Figure 4.1, if we suppose that the four annotated visual relationships represent *all* of the visual relationships annotated for the image, then, recall@N (for $N > 3$) for the baseline deep learning system is $1/4 = 0.25$, and recall@N for the neurosymbolic system (that leverages OWL-based knowledge graph reasoning) is $3/4 = 0.75$.

Two regimes of experimentation In Lu et al. (2016), the paper which introduced the VRD dataset, the authors distinguish between what they call a 'visual relationship detection' regime of experiments, and a 'predicate detection' regime of experiments. This set a pattern which others have followed, including ourselves. In the 'visual relationship detection' regime, one uses the objects detected by an object detector for training any downstream components of a visual relationship detection system. In the 'predicate detection' regime, one uses the objects in the annotated visual relationships for training downstream components. The 'predicate detection' regime is akin to simulating the existence of an object detector—a perfect one that detects precisely those objects that have been annotated. It eliminates the noise of the object detection, and puts all the attention on the prediction of predicates.

The performance of our object detector was indeed noisy. With the 'predicate detection' regime of experimentation, however, we were much more confident that the changes in metric scores we observed in our experiments were due exclusively to the effects of OWL-based knowledge graph reasoning. We did not have to worry that perhaps the signal we were receiving had been distorted by noise. We came to regard the 'predicate detection' regime of experimentation as 'noise cancellation mode', and eventually we came to use it exclusively. Our true subject of study is exploring ways of combining subsymbolic learning with symbolic OWL reasoning, and examining the effects or benefits of doing so. For us, the computer vision task of visual relationship detection in images is merely a context (a somewhat arbitrary context) in which to study our subject. We had no qualms when we eventually effectively abandoned the 'visual relationship detection' regime of experimentation in favour of the 'predicate detection' regime. We did not wish to study our subject through a cloudy microscope.

4.2 Our baseline deep learning visual relationship detection system

Here we discuss the architecture of our baseline deep learning visual relationship detection system. We begin with a high-level overview. Then we discuss components of the architecture in detail, and describe how we worked with them for the purposes of our three investigations.

4.2.1 Overview

Figure 4.2 shows the architecture of our baseline deep learning system for detecting visual relationships in the VRD images of the NeSy4VRD dataset. Conceptually, the system is a simple pipeline of two components: an object detection neural network followed by a predicate prediction neural network. The object detection neural network detects objects in the VRD images, localising them with bounding boxes, and classifying the object within each bounding box. The predicate prediction neural network (PPNN) predicts relationships between *ordered* pairs of the objects in an image. Figure 4.2 also shows how the noise of object detection (due to misidentified objects) can be cancelled by working with the annotated objects in place of detected objects.

4.2.2 Object detection

Here we describe our object detection model and how we worked with it. The model we used for object detection is the PyTorch implementation of a *Faster R-CNN ResNet50 FPN* object detector. PyTorch provides a pretrained instance of the model that is trained

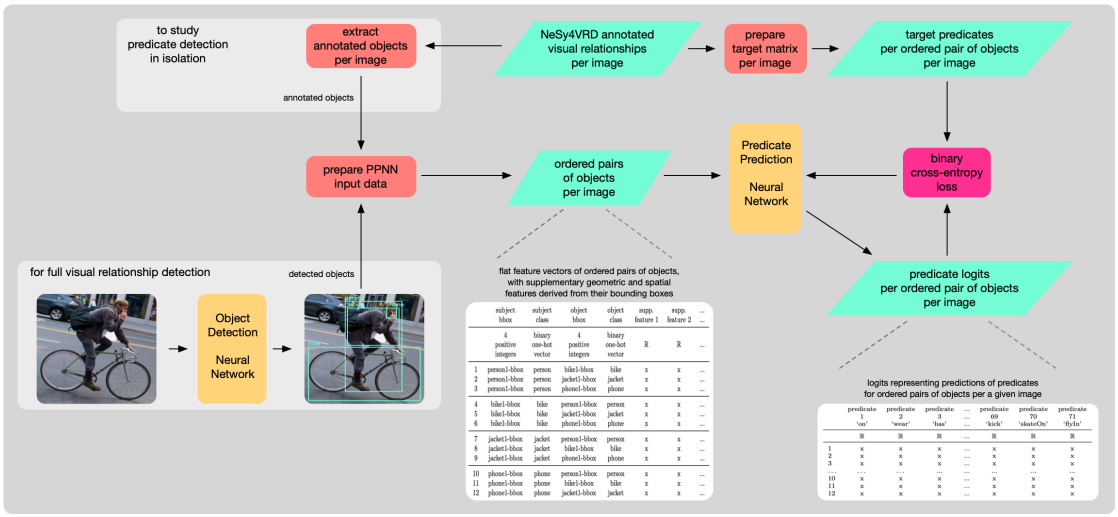


Figure 4.2: Our baseline deep learning system for detecting visual relationships. An object detection neural network detects objects in images, and a multi-label predicate prediction neural network predicts relationships between ordered pairs of those objects. To study the predicate detection aspect of visual relationship detection in isolation, the noise of the object detection can be cancelled by using the objects annotated for the images in place of those detected by an object detector.

on 90 of the object classes present in the images of the COCO (Common Objects in Context) dataset (Lin et al., 2014). These 90 COCO object classes overlapped strongly with the 109 NeSy4VRD object classes, so we adopted a transfer learning approach to speed the training of this large model. We adapted the two output layers of the model to fit NeSy4VRD, and we configured the model to keep all five layers of the ResNet50 backbone frozen, because we found this delivered better object detection performance.

The Faster R-CNN model is large and complex. A single checkpoint file of our customised version consumes 0.5GB of disk space. The model overwhelmed our consumer computing hardware. In the end, we trained it on an Nvidia A100 GPU, using the Hyperion high-performance computing cluster at City St George's, University of London.

For loss calculation, there was nothing to do because the Faster R-CNN model computes its own loss, internally. In fact, when the model is called during training, it returns nothing but the four components of loss it computes. There is no evidence of what the model is learning to predict. The user's only option is to sum the four components of loss and invoke backpropagation on the sum.

For evaluating the performance of the object detector, our primary metric was recall, because the images of NeSy4VRD are not annotated exhaustively, which makes calculating precision infeasible. But using recall alone was insufficient for our needs because

we wanted to manage the mean number of objects detected per image. We did not want our PPNN (our predicate prediction neural network) to be fed huge numbers of ordered pairs of objects, and to thereby predict huge numbers of visual relationships, because our ultimate metric for measuring visual relationship detection predictive performance is recall@N, as explained above. We did not want to invite serious *crowding-out* of hits, because this might well impact negatively on our ability to detect the effects of OWL reasoning in observed recall@N results for our main values of N, (25, 50, 100).

Thus, in addition to considering recall in performance evaluation, we also considered the mean number of objects detected per image. We sought a mean number of detected objects per image that was a bit larger than the mean number of annotated objects per image, but not too much larger. Thus, ultimately, our criteria for evaluating the performance of our Faster R-CNN object detector, and for choosing a preferred model, were somewhat heuristic: we looked for the model with the best recall and what we judged to be a reasonable mean number of detected objects per image. During training, we saved model checkpoints every 10 epochs. We evaluated the performance of each model checkpoint on the NeSy4VRD validation set, and selected our preferred models using the criteria just described. The epoch number of the selected model was 250.

The Faster R-CNN ResNet50 FPN object detection model has many hyperparameters for tuning performance during inference. Thus, the selection of a preferred trained model was followed by extensive inference hyperparameter tuning trials. These trials equated to a grid search with respect to 4 particular inference hyperparameters upon which we focused our attention. To allow for the heuristic nature of our model selection strategy, we selected 4 different inference hyperparameter setting configurations for performing inference on the VRD images of NeSy4VRD. The four hyperparameter setting configurations delivered different levels of recall for different values of the mean number of detected objects per image. The largest recall delivered by our four configurations was 0.6.

These respectable but modest object detection recall results gave us cause for concern. Having observed them, we began to worry that we would find the 'visual relationship detection' regime of experimentation, described in Section 4.1, which accepts the noise of object detection, to be problematic for our study of the effects of OWL reasoning on visual relationship detection predictive performance.

We performed inference on the NeSy4VRD test set, using our single trained object detector, four times, once for each of our 4 inference hyperparameter setting configurations. The outputs (the objects detected for each test set image) were stored on disk and later converted into PPNN input data, ready for use in training PPNN models.

4.2.3 Predicate prediction

Here we describe the predicate prediction neural network (the PPNN) of our baseline deep learning visual relationship detection system, and how we worked with it.

The PPNN model Our PPNN model is a small feed-forward, multi-class, multi-label classifier that accepts flat feature vectors as input. Its job is to predict predicates that describe visual relationships between ordered pairs of objects. We gave our PPNN model four linear hidden layers, each of which uses an ELU activation function, and one linear output layer with no activation function, meaning the model returns raw logits. The first three hidden layers are twice the size of the input feature vectors, and the 4th hidden layer is the size of the inputs. The size of the output layer, 71, matches the number of predicates defined in NeSy4VRD. In recognition of the fact that the PPNN is a multi-label classifier, when we convert logits to probabilities, we do so using a sigmoid function, not a softmax.

When we chose this architecture for our PPNN, our objective was to design a moderately capable learner. We sought an architecture that would enable the PPNN to evidence reasonable learning capability on its own, but not *too* effective. Our hypothesis was that, if the PPNN is too capable of learning on its own, there would be no 'head room' for OWL reasoning to show its potential for guiding the PPNN's learning, and for uplifting its predictive performance. Happily, our choice of PPNN architecture proved to be a good one in this regard.

The reason our PPNN model is a multi-label classifier is that it is common for the NeSy4VRD annotated visual relationships for an image (*i.e.*, an image's scene graph), to use two, three or more predicates in relation to a single ordered pair of objects. For example, here are two representative patterns that occur:

- `(person wear Y)`, `(person has Y)`, `(person in Y)`, where `Y` is some wearable thing, like a `shirt`, `jacket`, `jeans`, etc.;
- `(X beside Y)`, `(X nextTo Y)`, `(X near Y)`, `(X by Y)`, where the predicates are synonyms or near synonyms of one another.

Our baseline visual relationship detection system has a hyperparameter to control the number of predicates that can be predicted for a single ordered pair of objects. For each of our three investigations, described in forthcoming sections, we gave this hyperparameter a value of 5.

PPNN input data The baseline deep learning system architecture in Figure 4.2 indicates the important step of preparing input (training) data for the PPNN. Input data for the PPNN are flat feature vectors. The core features in a PPNN input feature vector

are an ordered pair of objects, where each object is represented by its bounding box (four integers) and a binary one-hot vector representing its class. To help the PPNN learn, we supplement these core input features with a small number of geometric features derived from the bounding boxes of the ordered pair of objects in question. We compute the sine and cosine of the angle between the centroids of the two bounding boxes, and we compute two inclusion ratios that measure the extent to which one bounding box is included (contained) within the other. The idea for supplementary geometric features such as these we borrowed from Donadello and Serafini (2019).

PPNN training To help decide the best platform to use for training our PPNN, we conducted a systematic platform benchmarking exercise that included our personal consumer hardware, the Hyperion HPC cluster at City St George's, University of London, and several AWS (Amazon Web Services) EC2 instance types. To our surprise, we discovered that our personal consumer hardware (Apple Mac machines with M2 Pro and M2 Ultra chips) proved to be the best options for speed and cost, and by some margin. A copy of the report we wrote to document this benchmarking exercise can be found in Appendix B.

For training the PPNN, we set the number of training epochs to 100, by default, but the early-stopping strategy we implemented always triggered somewhere in the range of 30 to 60 epochs. The metric we tracked for early-stopping purposes was the performance of the model under training on the NeSy4VRD validation set, as measured by recall@50. Our early-stopping patience value was 25. We justified this seemingly large value for patience to ourselves on the basis that we saved a PPNN model checkpoint at every epoch, not at epoch intervals. We justified it further by the fact that we were using recall@50 as a proxy for our two other core metrics, recall@25 and recall@100, and that therefore a generous patience value would better ensure we always identified the very best performing model, for each of our different recall@N metrics. We computed PPNN training loss (for the baseline deep learning system) using standard binary cross-entropy only.

PPNN inference and prediction selection Our policy for performing PPNN inference (on the NeSy4VRD test set) was to process the 35 latest PPNN model checkpoint files (*i.e.*, the 35 model checkpoints with the largest epoch numbers). We decided to endure the extra processing time entailed by this decision in order to better ensure that we found the best performing models for each of our four recall@N metrics, where $N \in \{25, 50, 100, 999\}$. The motivation was not to maximise reported recall@N scores, but to help ensure that if OWL reasoning did have an effect on recall@N (positive or negative), that we find that full effect, and not some approximation of the full effect. During inference, the PPNN model was allowed to generate as many predicted visual relationships for each image as it wanted. The predictions (inference outputs) of the

model checkpoint files were stored on disk to allow them to be processed separately, and perhaps repeatedly, in the context of different hyperparameter settings.

A 'prediction selection' step in the overall PPNN training/inference/evaluation pipeline then selected from amongst the predicted visual relationships for an image according to two hyperparameters: the confidence of the prediction (as determined by the sigmoid of the PPNN logit that corresponds to a particular predicate), and the maximum number of predicates allowed per ordered pair of objects. We found that our PPNN model was rather parsimonious with respect to the number of visual relationship predictions that it tended to generate per image, so we gave relaxed settings to these two hyperparameters. For the minimum confidence of a prediction we used 0.5, and for the maximum number of predictions per ordered pair of objects we used 5. The predictions selected in this step of the overall PPNN pipeline were themselves stored on disk, for each of the (up to) 35 model checkpoints being evaluated.

PPNN performance evaluation To evaluate performance, we processed the selected predictions for each of the (up to) 35 model checkpoints four times, once for each value of recall@N, where $N \in \{25, 50, 100, 999\}$. As well as computing recall@N for each image, and computing mean recall@N over the images, this evaluation step also updated the selected predictions as they were being processed. The updates applied to the selected predictions, for a given value of N in recall@N, recorded whether or not a given predicted visual relationship had been *submitted* for performance evaluation, and, if so, whether or not it turned out to be *hit*. This facilitated later analysis of the performance of our PPNN models at the granular level of individual predicted visual relationships, and it proved to be valuable in that regard.

PPNN model selection A subsequent step of the overall PPNN pipeline then gathered the model checkpoint-specific performance results, for a given N in recall@N, into a single '.csv' file, and, at the same time, marked in that '.csv' file the epoch (model checkpoint) that delivered the maximum recall@N score. A subsequent step of the PPNN pipeline then re-processed these recall@N-specific '.csv' results files in order to extract the *epoch* of the best performing model, along with its corresponding (maximum) recall@N score. We captured the epoch of the best performing model (for each value of N) as an approximate way of measuring and comparing the *speed* of subsymbolic learning.

PPNN pipeline automation We automated the execution of our PPNN pipeline with a shell script that invokes each step of the pipeline in sequence. All communication between steps is via files on disk. The outputs of one step are the inputs to the next. Everything was preserved for every experiment.

Summary The design of our multi-step PPNN pipeline enabled us to analyse the effects of OWL reasoning on the subsymbolic learning of our PPNN models from three perspectives: *(i)* the effects on predictive performance as measured by recall@N, *(ii)* the effects on predictive performance as reflected by the individual predicted visual relationships, and *(iii)* the effects on the speed of subsymbolic learning. And our model selection strategy helped to ensure that we looked for those effects amongst the very best performing models.

4.3 Metrics

Here we describe the metrics that we use for evaluating the predictive performance of our baseline and neurosymbolic visual relationship detection systems, and for evaluating the effects of symbolic OWL reasoning on the deep (subsymbolic) learning more generally.

recall@N Beginning with the paper that first introduced the VRD dataset (Lu et al., 2016), researchers using it have focused virtually exclusively on measuring recall@50 and recall@100 to compare the performance of their respective visual relationship detection systems. The reason for the focus on recall is that the images of the VRD dataset (and, similarly, of our NeSy4VRD dataset) are *not annotated exhaustively*, either in terms of objects or relationships. Exhaustive manual annotation is infeasible. The consequence is that counts of false positive visual relationships are exaggerated, because perfectly good predictions, of things that simply were not annotated (but might well have been), are treated as false positives. This leads to precision-related metric scores that can be grossly pessimistic and misleading. We continue the convention of measuring recall@50 and recall@100, and for the same reasons, but we expand this set of metrics to include recall@25 and recall@999 to help us discern effects of OWL reasoning on the subsymbolic learning of our PPNN—which is our prime objective (not visual relationship detection for its own sake). More precisely, the recall@N metric that we use is 'mean per image recall@N'. That is, we first calculate recall@N scores for each test set image, and then we take the average. Per image recall@N is simply the number of hits (within the topN most confidently predicted VRs) over the number of ground-truth (annotated) VRs. We use $N = 999$ as a tactic to allow us to measure performance using all of the predicted visual relationships for a given image, not just the N most confidently predicted visual relationships. This helps us to detect when predicted visual relationships that are *hits* are being *crowded-out* from the three other 'topN' categories because the confidence attached to those hit predictions is too low.

mAP@N Despite the challenges associated with precision-related metrics just described, we opted to measure mAP@N (mean average precision), and for the same val-

ues $N \in \{25, 50, 100, 999\}$ as for recall@N. The justification is that, while the mAP@N scores will not be meaningful in absolute terms, the changes in the neurosymbolic system mAP@N scores, relative to baseline, and relative to one another, can still reveal useful insights into the effects of OWL reasoning. First, we compute average precision (AP) for each test set image, using

$$AP = \sum_{k=1}^{n} p(k)\Delta r(k) = \sum_{k=1}^{n} p(k)\Big(r(k) - r(k-1)\Big),$$

where k is the rank in a set of predicted visual relationships sorted in descending order by confidence, n is the number of predicted visual relationships, $p(k)$ is the precision at cut-off k, and $\Delta r(k)$ is the change in recall from items $k-1$ to k. Since $r(0)$ is not defined, we set $r(0) = 0$. Then we average the per-image AP scores to get mAP@N, using

$$mAP = \frac{\sum_{t=1}^{T} AP(t)}{T},$$

where T is the number of NeSy4VRD test set images.

Precision, adjusted precision, and 'false false positve rate' To get truer estimates of precision (in absolute terms) than mAP@N could provide, we measured precision manually, via visual inspection. We randomly selected a small subset (25) of the test set images, from amongst those having more than 8 but less than 25 predicted visual relationships. This latter condition helped keep this manual exercise tractable, and it had the added benefit that the scores calculated correspond simultaneously to all values of $N \in \{25, 50, 100, 999\}$ for a hypothetical precision@N metric. We displayed each predicted visual relationship, individually, against its associated image, and simultaneously observed whether or not it had been marked as a *hit* or not. It was soon clear that the majority of predictions not marked as hits were, in fact, perfectly good predictions of visual relationships that might well have been annotated, but simply had not been. So each predicted visual relationship not marked as a hit (*i.e.*, each ostensible false positive, or FP) was further classified as being either a true false positive (TFP) or a false false positive (FFP), giving, for each image, $FP = TFP + FFP$, and where:

- TFP = true false positive (*i.e.*, definitely a bad prediction)
- FFP = false false positive (*i.e.*, either definitely good or a borderline case).

This factorisation (or classification) of FPs permitted us to compute two measures of precision: the standard one, and an adjusted one that excludes the FFPs. Thus, standard precision (per image) was measured using

$$precision = \frac{TP}{TP + FP} = \frac{TP}{TP + (TFP + FFP)},$$

and *adjusted* precision (per image) was measured using

$$precision_{adj} = \frac{TP}{TP + TFP}.$$

The classification of the FPs also permitted us to devise a further metric that we call the 'false false positive rate', or FFP rate, for short. This measures the proportion of FPs (the non-hits) that, upon visual inspection, were deemed to be good predictions of visual relationships. The FFP rate is defined as

$$FFPrate = \frac{FFP}{FP} = \frac{FFP}{TFP + FFP}.$$

Learning speed For investigation 1 (annotation augmentation), we evaluate the learning speed of the PPNN, measured in training epochs. In Section 4.2.3, we explained that every epoch we save a model checkpoint. The first model checkpoint (*i.e.*, the lowest training epoch) to deliver the maximum observed recall@N score, for $N \in \{25, 50, 100, 999\}$, is selected for evaluation. We treat the epoch numbers of these ($topN$-specific) 'earliest best performing models' as measures of PPNN (subsymbolic) learning speed.

Incidence rate of semantically invalid predictions The primary metric for evaluating OWL reasoning in investigation 2 (using prediction validation to apply logical constraints) is the incidence rate of semantically invalid predicted visual relationships. The OWL reasoning strategy employed in investigation 2 was conceived to reduce this rate. We measure this incidence rate by counting the individual instances of predicted visual relationships that are found to be semantically invalid.

Volume of predictions per image We also examine the mean number of predicted visual relationships per image. This metric yields findings that are interesting in themselves, and that help us better understand the recall@N and mAP@N scores that we observe.

Symmetric pairs and inverse pairs Finally, investigation 1 (annotation augmentation), we also do a deep dive into the effects of two particular subcategories of OWL link inference semantics: symmetry and inverses. We devised several metrics to help us probe the extent to which *symmetry* inferences being activated leads to the PPNN learning to better predict visual relationships in symmetric pairs. We measure things such as *(a)* the mean number of predicted visual relationships (VRs) where the predicate is a symmetric property (per VRD-World), *(b)* the mean number of predicted VRs that belong to a symmetric pair (regardless of topN rank), *(c)* the mean number of

predicted VRs that belong to a symmetric pair, where the pair is within the same topN rank of predictions, and *(d)* the mean number of hits where the predicate is symmetric. We also devised counterpart metrics to help us probe the extent to which *inverse* inferences being activated leads to the PPNN learning to better predict visual relationships in inverse pairs.

4.4 Knowledge graph reasoning for annotation augmentation

In this section we describe investigation 1 of thread two of our research into combining symbolic OWL reasoning with subsymbolic learning. We start with an overview of the investigation and a description of relevant methods. Then we present results and discuss them.

4.4.1 Overview and methods

In investigation 1 we use an OWL-based knowledge graph tool, hosting our VRD-World OWL ontology (from NeSy4VRD), and OWL reasoning, in the guise of a symbolic reasoning engine, to augment the visual relationships that have been annotated for each image. If we look at this idea from the perspective of scene graphs, we can say that we use the same technologies to *materialise the scene graph* of each image. The augmentation (materialisation) infers those visual relationships that are logically entailed by the annotated visual relationships, in light of the domain description declared in the VRD-World ontology hosted in the knowledge graph tool.

The reason we perform the augmentation (materialisation) is that the NeSy4VRD visual relationships that have been annotated for each image are *sparse*, rather *arbitrary*, and *inconsistent*. In a typical image, only a handful of the potential visual relationships have been annotated. The choices that annotators have made (what to annotate, what not), appear random. And from one image to the next, inconsistencies in the annotations doubtless result in contradictory loss signals being conveyed during neural network training.

Consider a simple example of inconsistency. Suppose that image X has an annotated visual relationship that says `(A beside B)`, and that the next image, image Y, has an annotated visual relationship that says `(C nextTo D)`. If the PPNN learns to predict `(A beside B)` for image X, it gets encouraged to continue doing so, by attracting a small loss contribution. But if it learns to predict `(C beside D)` for image Y, it gets penalised with a large loss contribution. It is reasonable to expect that any neural network, such as our PPNN model, would struggle to learn efficiently and effectively under such

conditions.

These conditions of thin and inconsistent supervision, however, that are a reality with respect to the 'baseline' NeSy4VRD visual relationship annotations, provide an opportunity for OWL reasoning to add value. Suppose that the version of the VRD-World OWL ontology hosted in the knowledge graph tool declares that these two properties, `beside` and `nextTo`, are equivalent, by asserting the triple `(beside owl:equivalentProperty nextTo)`. In OWL, if two properties are equivalent, then whenever one is used, the other can be inferred. Thus, after OWL reasoning is performed, the materialised scene graph for image X will contain this pair of triples: `(A beside B)` and `(A nextTo B)`. And the materialised scene graph for image Y will contain this pair: `(C nextTo D)`, `(C beside D)`.

Thus, the hypothesis that underlies investigation 1 is that, by augmenting the annotated visual relationships according to the VRD-World OWL ontology, and thereby enriching and making more consistent the supervision the annotations provide during training, OWL reasoning can impact the subsymbolic learning of the PPNN model positively. We would hope to see evidence of this positive impact of OWL reasoning in the three areas mentioned earlier: *(i)* the predictive performance as measured by recall@N, *(ii)* the predictive performance as reflected by the individual predicted visual relationships themselves, and *(iii)* the speed of PPNN learning, as measured by the epochs of the best performing PPNN models.

Figure 4.3 summarises investigation 1 pictorially. As the figure implies, we used OWL reasoning to perform the augmentation of the annotated visual relationships *offline*, in advance of training any PPNNs. We could have opted instead to use our symbolic reasoning engine (our OWL-based knowledge graph hosting our VRD-World ontology) in *real-time*, as the PPNN is being trained. We show that this works in investigations 2 and 3. Both options work functionally. But it made sense, in our case, to do things offline, once, because OWL reasoning, like logical inference generally, is deterministic. The materialised scene graph of image X will never change, no matter how many times it is inferred (as it comes round in successive epochs). Training may well be slower if the symbolic reasoning engine is used in real-time (depending on the knowledge graph tool selected for use). But use cases different to ours surely exist, where the inference to be performed will never be knowingly redundant, and hence real-time OWL reasoning is the appropriate choice. Investigations 2 and 3 have this character.

Sub-categories of link inference, and versions of VRD-World Here, in investigation 1, we exercise a particular category of OWL reasoning. We call this category 'link inference', where new triples are inferred that link existing individuals in new ways. The top half of Figure 4.3 illustrates link inference. The inference semantics of the full VRD-World ontology exercise five different *sub-categories* of link inference.

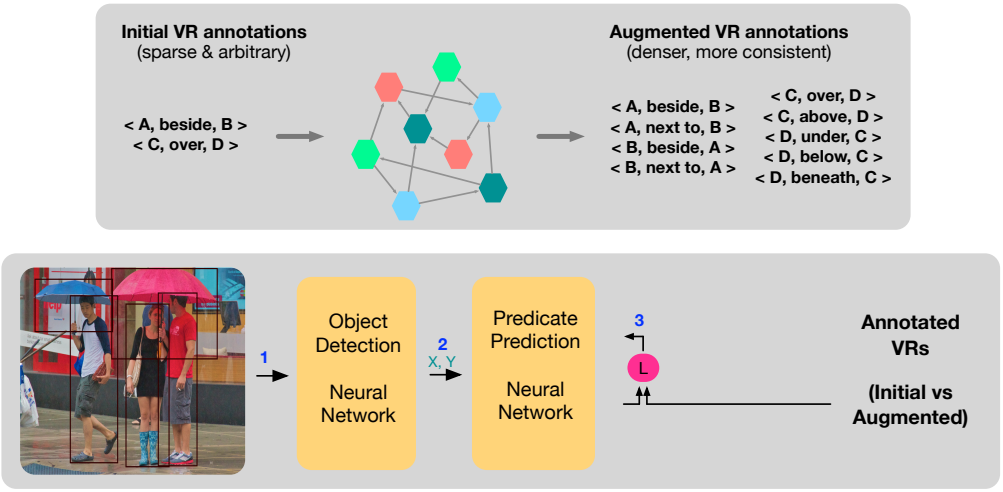


Figure 4.3: The setting for investigation 1 of thread two. The top half of the figure shows an OWL-based knowledge graph augmenting the annotated visual relationships of some hypothetical image. The presumption is that this knowledge graph hosts our VRD-World ontology. The bottom half of the figure shows the annotated visual relationships being used to provide supervision during PPNN training. A baseline deep learning system is trained using only the 'initial' sparse and arbitrary visual relationships for supervision. A neurosymbolic system is trained using the 'augmented' visual relationships that provide enriched supervision.

These sub-categories have to do with symmetry, inverses, transitivity, equivalence, and sub-properties. Some of the user-defined properties of VRD-World (which correspond to the NeSy4VRD predicates, one-to-one) are declared to be symmetric, (*e.g.*, `beside`). Some are declared to be transitive (*e.g.*, `above`). Some have inverses (*e.g.*, `over` and `under`), or are semantically equivalent to one another (*e.g.*, `beside` and `nextTo`). And many are sub-properties of some other property (*e.g.*, `wear` is a declared to be a sub-property of `has`). For investigation 1, we examined the effects of OWL reasoning on PPNN subsymbolic learning for each of these five sub-categories of link inference, individually, and in combinations.

To allow for that examination, we created multiple different *versions* of our VRD-World OWL ontology, each one possessing a different mix of the five link inference capabilities. Each version materialises the scene graphs of the images in slightly different ways. Hence, each version leads to a different neurosymbolic visual relationship detection system, that performs differently from the others. We refer to these different versions of the VRD-World ontology, and to the different neurosymbolic systems that they give rise to, as neurosymbolic *treatments*. We assign names to treatments so we can talk about them, like 't01' and 't02', where 't' denotes 'treatment'. When we say 'baseline', we refer to the baseline deep learning system, where the PPNN was trained on the baseline NeSy4VRD annotated visual relationships, and under the 'predicate detection' regime

Table 4.1: The meaning of the names for the treatments used to explore the effects of OWL reasoning strategy 1 (annotation augmentation) on subsymbolic learning. For OWL reasoning strategy 1, treatments are configurations of activated link inference semantics within OWL ontology VRD-World, and the corresponding neural network (PPNN) models that emerge from having been trained on target scene graphs logically enriched by OWL reasoning governed by that configuration of VRD-World.

treatment name	treatment key	mixture of activated OWL link inference semantics	relevant Semantic Web (OWL) construct
t01	S____	symmetric properties	owl:SymmetricProperty
t02	_I___	inverse relationships	owl:inverseOf
t03	__T__	transitive properties	owl:TransitiveProperty
t04	___E_	equivalence relationships	owl:equivalentProperty
t05	____U	subproperty relationships	rdfs:subPropertyOf
t07	___EU	t04, t05	
t08	__TEU	t03, t04, t05	
t09	S_TEU	t01, t03, t04, t05	
t10	SITEU	t01, t02, t03, t04, t05	
t11	_ITEU	t02, t03, t04, t05	
t12	SI___	t01, t02	
t13	_IT__	t02, t03	
t14	S__E_	t01, t04	

of experimentation (explained earlier), where the noise introduced by object detection has been cancelled. Here in investigation 1, when we refer to a treatment (*e.g.*, 't01'), we refer to a particular neurosymbolic system, where the PPNN was trained on annotations that were augmented by OWL reasoning, using a particular version of the VRD-World ontology, with its particular mix of link inference semantics, under the 'predicate detection' regime of experimentation. Table 4.1 defines the various treatments we explored in connection with the OWL reasoning strategy employed in investigation 1 (annotation augmentation).

At all times, whether for the baseline or for a treatment, we trained and evaluated four instances of the system. Thus, the summary results we present have underlying sample sizes of four.

4.4.2 The effects of OWL reasoning on recall@N (A)

We examine the effects of OWL reasoning on recall@N in two different settings, (A) and (B). In this section, we consider setting (A). Setting (B) is considered in the section which follows. In setting (A), for each treatment system, inference was performed on the *baseline* NeSy4VRD test set annotations. In other words, the treatment systems were trained on augmented annotations, but were evaluated on baseline annotations. The hypothesis is that setting (A) (*i.e.*, this experiment design) gives OWL reasoning

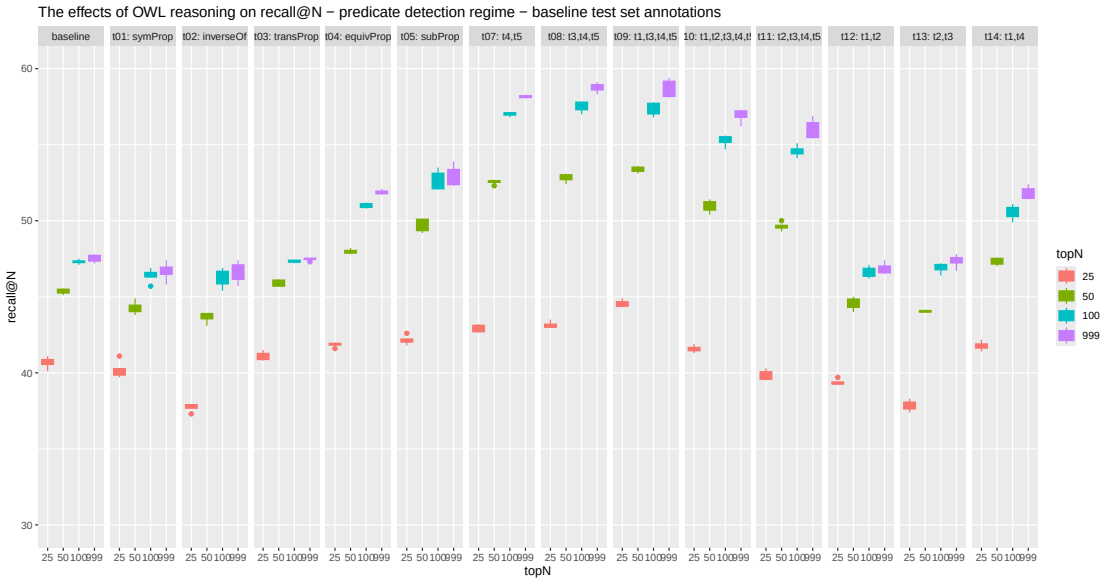


Figure 4.4: The effects of OWL reasoning on recall@N, in the predicate detection regime of experimentation (where the noise of object detection is cancelled), and when performance is evaluated against the baseline NeSy4VRD test set annotated visual relationships.

its best opportunity to show evidence of its effects on subsymbolic learning. Figure 4.4 shows the effects of OWL reasoning on recall@N in setting (A).

Figure 4.4 shows that 'link inference' OWL reasoning, governed by our VRD-World ontology, provokes a variety of responses in the subsymbolic learning of our PPNN. Treatments like t01 (symmetry) and t02 (inverses) deliver recall@N slightly lower than baseline. Treatment t03 (transitivity) delivers recall@N similar to baseline. Treatments t04 (equivalence) and t05 (subProperties) deliver recall@N much higher than baseline, particularly for values of N greater than 25. Combination treatments, like t07, t08, t09, and t10, where sub-categories of link inference are exercised jointly by OWL reasoning, deliver recall@N far higher than baseline.

The strong recall@N responses delivered by combination treatments suggests co-operative interaction between the different sub-categories of link inference semantics. Sometimes, the co-operative interaction appears additive. For example, consider treatments t04, t05, and t07. For recall@100, the uplift from baseline for t04 is about 4 percentage points (47% to 51%); the uplift for t05 is about 5.5 percentage points; and the uplift for t07, which combines t04 and t05) is about 10.

Sometimes the interactions suggest mutual reinforcement and amplification rather than additivity. For example, consider treatments t01, t08 and t09. Treatment t01 (symmetry, on its own), appears to suppress recall@N. But in t09, which symmetry with t08, we

see that the introduction of the inference semantics for symmetry has a positive effect on recall@25, and to a lesser extent on recall@50.

Figure 4.4 reveals that link inference OWL reasoning effects the predictive *precision* of the visual relationship detection systems as well as their *recall*. For example, observe that recall@50 for treatment t04 *exceeds* recall@100 (and recall@999, which is similar) for the baseline system. This represents at least a doubling of precision. Recall@50 for treatment t09, which exceeds baseline recall@999 by a wide margin, therefore, represents a far greater increase in precision.

4.4.3 The effects of OWL reasoning on recall@N (B)

Here we examine the effects of OWL reasoning on recall@N for setting (B). In setting (B), instead of evaluating each treatment system on the baseline NeSy4VRD test set annotations, the systems were evaluated on augmented test set annotations. For each treatment system, the augmentation of the test set annotations, performed by OWL reasoning, was governed by the same version of VRD-World that governed the augmentation of the training set annotations. The hypothesis for setting (B) is that, by augmenting the test set annotations, the visual relationship detection task becomes harder, because now there are many, many more ground-truth annotations (bigger scene graphs) for each test set image. Thus, intuition suggests that recall@N is likely to fall, perhaps significantly. Figure 4.5 shows the effects of OWL reasoning on recall@N in setting (B).

The main thing to notice about Figure 4.5 is how similar it is to Figure 4.4, not just in overall shape but in the absolute values of recall@N being reported. Surprisingly little has changed. The neurosymbolic systems (in most cases, but not all) perform as well as they did in setting (A), where the visual relationship detection task was much easier. This is a counter-intuitive result. It appears to indicate that the (subsymbolic) PPNN is capable of learning the regularities of the link inference semantics of OWL remarkably well.

4.4.4 The effects of OWL reasoning on the volume of predictions

Here we examine the effects of OWL reasoning on the distribution of the number of predicted visual relationships per image. Figure 4.6 compares treatment t09 with the baseline, in this regard. We can see that treatment t09 has a powerful impact on the distribution of the number of visual relationships that get predicted per image. Treatment t09 induces the PPNN to generate many more predictions than baseline. This helps to explain the uplift in recall@N delivered by treatment t09. It indicates that the enriched scene graphs produced by OWL reasoning help the PPNN to learn.

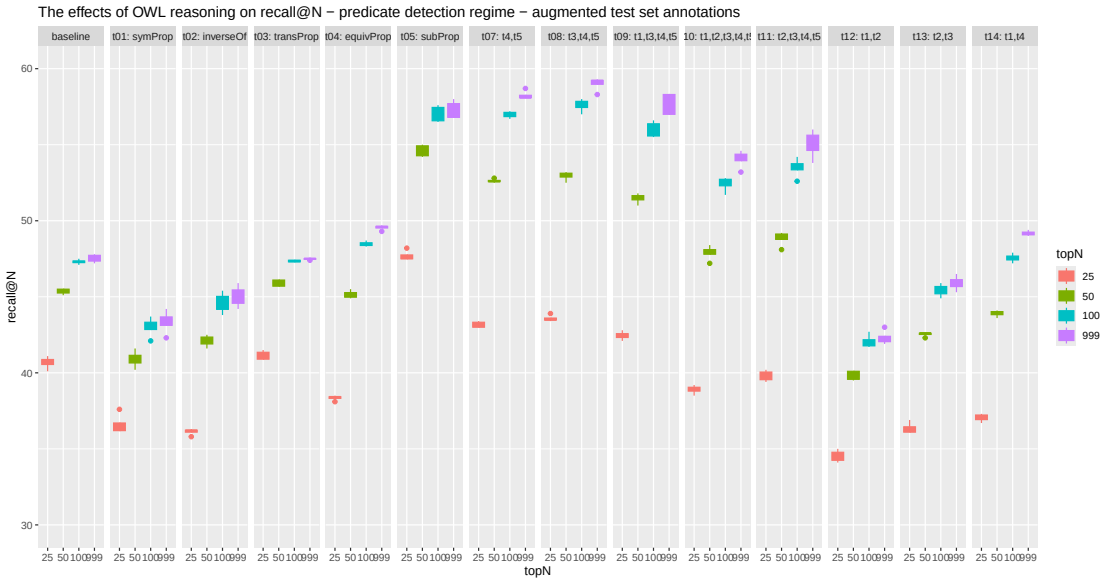


Figure 4.5: The effects of OWL reasoning on recall@N, in the predicate detection regime of experimentation, when performance is evaluated against NeSy4VRD test set visual relationship annotations that have augmented by OWL reasoning using the same version of the VRD-World ontology as was used to augment the training set visual relationship annotations upon which each model is trained.

4.4.5 The effects of OWL reasoning on mAP@N

Figure 4.7 shows the effects of using the annotation augmentation OWL reasoning strategy on mAP@N scores, for most of the same neurosymbolic treatments as in preceding results figures. Most of the neurosymbolic treatment systems underperform the baseline deep learning system with respect to mAP@N. The decreases in the mAP@N scores are, however, well explained by corresponding increases (relative to baseline) in treatment system prediction volumes, as seen earlier, in Figure 4.6. Once the increases in underlying prediction volumes are taken into account, the scale of the reductions in mAP@N relative to baseline look remarkably *small*. For example, consider neurosymbolic treatment t09 (S_TEU) relative to baseline. Figure 4.6 shows that the baseline deep learning system generates (for topN=999) about 33 visual relationship predictions per image, on average, whereas treatment t09 generates about 59—a relative increase in prediction volumes of 79%. Figure 4.7 shows that the baseline system's mAP@999 score is about 24%, whereas the score for treatment t09 falls to 22.5%—a relative decrease of just 6%. Thus, the key observation is not that mAP@N falls for the neurosymbolic treatment systems, but how well it *endures*, given the large increases in underlying prediction volumes induced by OWL reasoning strategy 1.

Another essential observation is that the *absolute values* of the mAP@N scores in Fig-

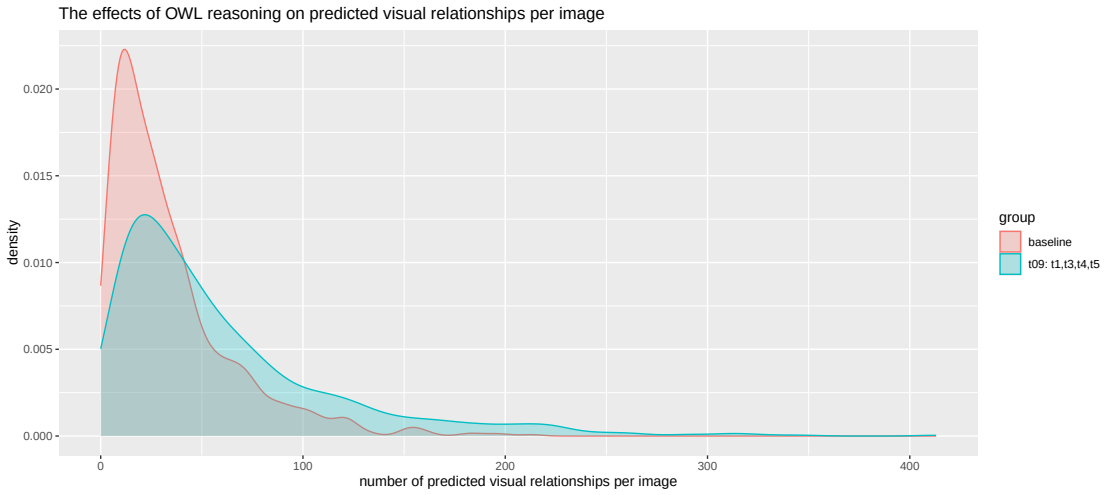


Figure 4.6: The effects of OWL reasoning on the distribution of the number of predicted visual relationships per image.

ure 4.7 are grossly pessimistic and misleading. Metrics like mAP (mean average precision) assume exhaustive annotation, which enables false positives to be identified reliably. But, as explained earlier, in Section 4.3, the images of NeSy4VRD are *not* annotated exhaustively, so false positives cannot be identified reliably, and they get overcounted as a consequence. That is, many predicted visual relationships that fail to match any annotated (target) visual relationship are perfectly good predictions of relationships that might have been annotated but were not. The mAP@N metric cannot distinguish these *false* false positives from the *true* false positives. Hence, when considering Figure 4.7, the useful information lies in the *relative changes* in the mAP@N scores, not their absolute values.

For example, Figure 4.7 gives helpful insight into the contributions of the OWL link inference semantics of inverses. Figure 4.7 shows that, for treatment t02 (_I___), mAP@N scores fall distinctively relative to baseline, yet Figure 4.6 shows no corresponding relative increase in prediction volumes to help explain this fall. We see a similarly distinctive relative fall in mAP@N between treatments t09 (S_TEU) and t10 (SITEU), yet Figure 4.6 shows that these treatments generate similar volumes of predictions. This evidence clearly associates the inference semantics of inverses with lower precision, as measured by mAP@N, at least.

To get a better sense of the precision delivered by our baseline deep learning and our neurosymbolic treatment systems in *absolute* terms, we measured precision manually during visual inspection of the individual visual relationships predicted for a random sample of 25 test set images. Table 4.2 reports the findings of this exercise, for the baseline deep learning system and the neurosymbolic treatment system t09 (S_TEU)—

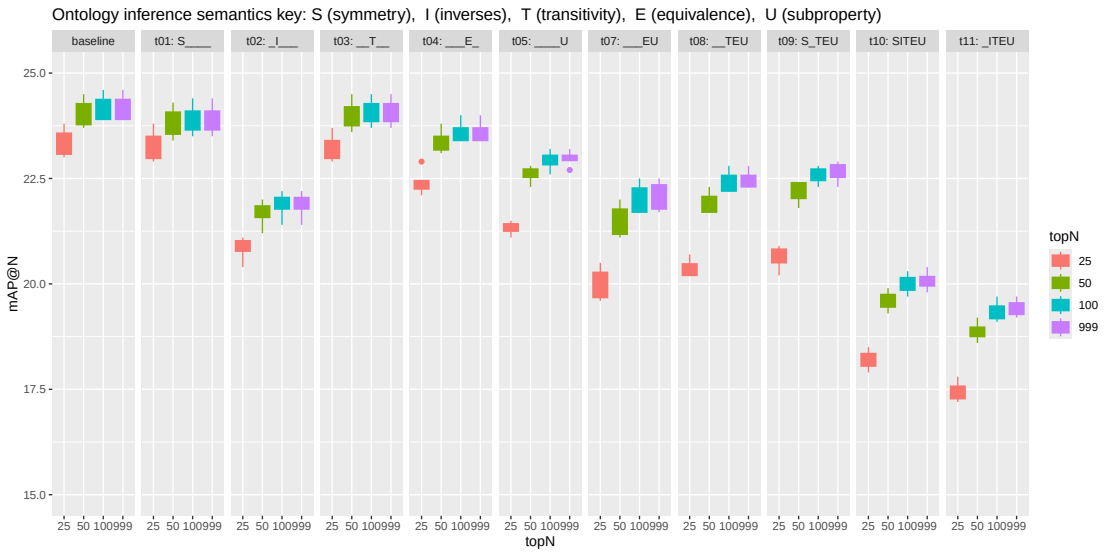


Figure 4.7: The effects of OWL reasoning on mAP@N. Different neurosymbolic treatments (combinations of OWL link inference semantics in OWL ontology VRD-World used to logically expand training image scene graphs) deliver mAP@N scores lower than a baseline deep learning system, to varying degrees. The salient observation, however, is how well mAP@N *endures*, given the corresponding, but far larger, relative increases in treatment prediction volumes induced by OWL reasoning.

the one that delivers maximal recall@N, per Figure 4.4. The standard precision scores are roughly in line with the mAP@N scores in Figure 4.7 and, like them, are grossly pessimistic and misleading. As explained earlier, in Section 4.3, our *adjusted* precision metric excludes *false* false positives (FFPs) (*i.e.*, good predictions of visual relationships not annotated). So the scores for 'adjusted precision' are the ones that provide truer estimates of precision in absolute terms. We see that adjusted precision for treament t09 falls relative to baseline, from a respectable 71.7% to 65.9%—a relative decrease of 8%. But given the relative increase in underlying prediction volumes of 79% (mentioned earlier), and given the relative increase of 53.8% reported in Table 4.2 for the random sample of 25 images, this 8% relative fall in adjusted precision is remarkably small.

Also explained earlier, in Section 4.3, the FFP rate reported in Table 4.2 gives the proportion of predicted visual relationships that fail to match an annotated (ground-truth) target that are, upon visual inspection, deemed to be good predictions of visual relationships that were simply not annotated. With respect to this metric, we see that the neurosymbolic treatment t09 (S_TEU) system outperforms the baseline deep learning system, lifting the FFP rate from 78.1% to 82.8%.

The increase in (standard) recall relative to baseline reported in Table 4.2 for treatment

Table 4.2: The effects of the annotation augmentation OWL reasoning strategy on precision, as measured manually via visual inspection of individual predicted visual relationships (VRs), for a small random sample of test set images. The adjusted precision scores exclude *false* false positives (FFPs) from consideration. The FFP rate gives the proportion of false positives deemed to be good predictions of visual relationships not annotated. Standard recall is hits per image over ground-truth annotations per image.

model	mean precision (standard) (%)	mean precision (adjusted) (%)	mean FFP rate (%)	mean predicted VRs per image (count)	% change predicted VRs per image (% chg)	mean recall (standard) (%)
baseline	30.2	**71.7**	78.1	11.9		47.4
t09 (S_TEU)	21.2	65.9	**82.8**	**18.4**	53.8	**50.9**

t09, from 47.4 to 50.9, is consistent with the moderate increases in recall@25 seen earlier, in Figure 4.4. Recall@25 is the only comparable metric in Figure 4.4 because, as explained earlier, in Section 4.3, the random sample of 25 images being analysed is drawn from the set of test set images for which the baseline system generated between 8 and 25 predictions, in order to keep the manual analysis tractable. Treatment t09 generated moderately more predictions per image—an average of 18.4 versus 11.9 for baseline. This is a healthy increase of 50%, but the volume of predictions still corresponds to a recall@25 metric only. For treatment t09, Figure 4.4 shows recall@25 increasing (relative to baseline) from 41 to 44. So the scale of the score increases for the two measures of recall (standard recall, and recall@25), relative to baseline, are consistent. The discrepancies in the absolute values are explained by the small size of our random sample.

4.4.6 The effects of OWL reasoning on hits per image

Here we examine the effects of OWL reasoning at a yet more granular level. We examine its effects on *(i)* the mean number of predicted visual relationships per image that are actually submitted for performance evaluation, and *(ii)* the mean number of hits per image. Figure 4.8 shows results for a selection of treatments.

As explained in Section 4.2.3, not all predictions that are generated for an image are actually submitted for evaluation. The predicted visual relationships selected for performance evaluation are those that meet certain criteria (*e.g.*, exceeding a minimum confidence threshold). This is what is reported in panel (A) for Figure 4.8. One observation with respect to panel (A) is that treatments t01 and t02 are submitting numbers of predictions equivalent to baseline, and yet, as we saw in Figures 4.4 and 4.5, treatments t01 and t02 deliver recall@N lower than baseline. This suggests a *weakening* in the *quality* of the predictions. Panel (B) of Figure 4.8 appears to confirm this. For treat-

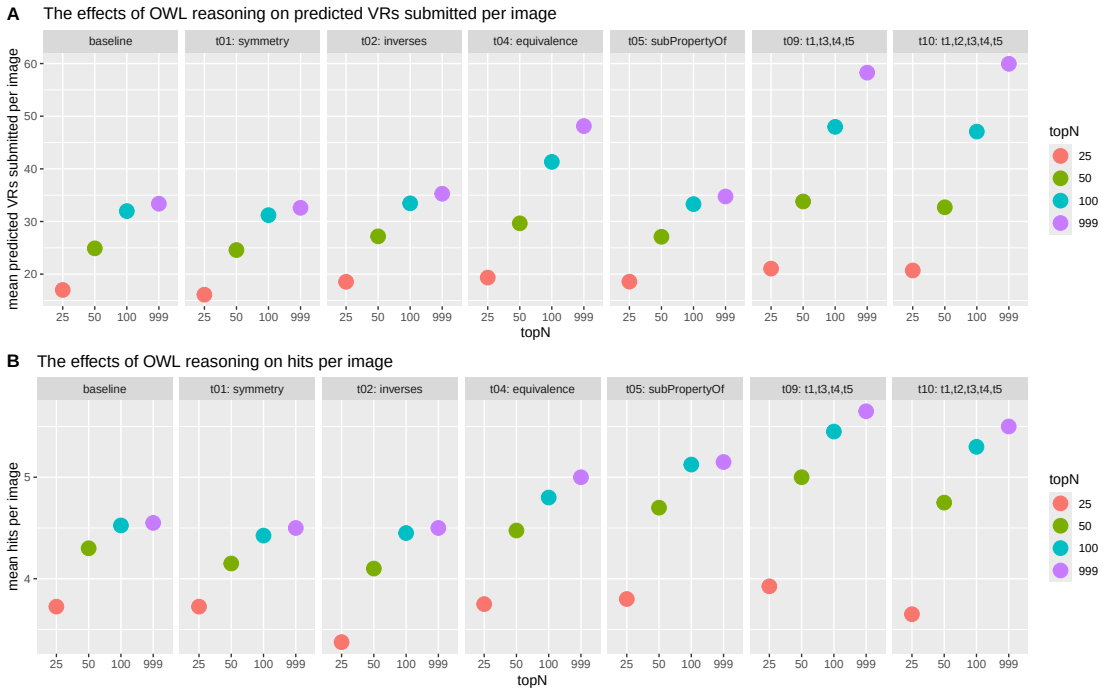


Figure 4.8: Panel A: the effects of OWL reasoning on the mean number of predicted visual relationships per image that are actually submitted for performance evaluation. Panel B: the effects of OWL reasoning on the mean number of hits per image.

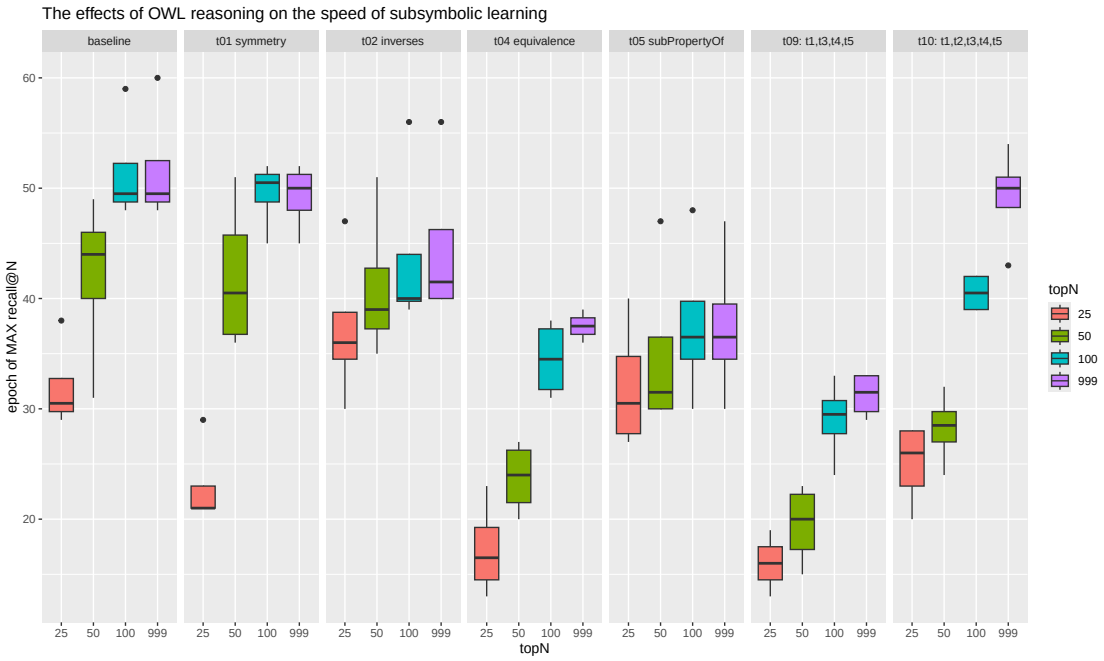


Figure 4.9: The effects of OWL reasoning on the speed of subsymbolic learning.

ments t01 and t02, the mean hits per image are lower than baseline, especially for topN values of 25 and 50. This appears to establish that symbolic OWL reasoning (for link inference) can effect not just the *quantity* of predictions generated by the subsymbolic PPNN, but the *quality* of those predictions as well.

We can see further evidence of this *quality* effect of OWL reasoning in Figure 4.8. Consider treatment t05. We can see that it submits the same number of predictions for evaluation as baseline, but the mean number of hits per image is much higher than baseline (relatively). This suggests a *strengthening* in the *quality* of the predictions.

4.4.7 The effects of OWL reasoning on learning speed

Here we examine the effects of OWL reasoning on the speed with which our (subsymbolic) PPNN learns. Recall that we captured the epoch numbers of the best performing models in order to use them as a heuristic way to compare speeds of learning. Figure 4.9 shows results for a selection of treatments.

Figure 4.9 indicates that symbolic OWL reasoning (for link inference) mostly increases the speed with which the subsymbolic PPNN learns, but not always. It depends on the mix of link inference semantics. Treatment t09, for example, relative to baseline, shows strong evidence of faster learning, for all topN values. Treatment t01 (symmetry, alone),

Table 4.3: Percentage changes in symmetric pairs relative to baseline for topN=50

Metric	t01	t09	t10	t14 (t1,t4)
(a) mean predicted VRs with a symmetric predicate (sp)	94	371	231	404
(b) mean predicted VRs with a sp occurring in pairs	286	885	559	982
(c) mean predicted VR sym pairs submitted together	279	475	332	707
(d) mean predicted VRs with a sp that become hits	18	183	150	184

on the other hand, delivers faster subsymbolic learning only for topN value 25.

4.4.8 The effects of OWL reasoning on symmetric pairs

Here we examine the effects of OWL reasoning with respect to particular sub-category of link inference semantics: the inference semantics of symmetry. For treatments that include the inference semantics of symmetry, we examine the frequency with which visual relationships predicted for symmetric predicates are predicted in symmetric pairs. In other words, we examine how well the subsymbolic PPNN learns which predicates are symmetric, and how well it learns to act on that knowledge, by predicting those predicates in symmetric pairs. Table 4.3 shows results from an analysis of the symmetric pairs present (or not) in the predicted visual relationships of a selection of treatments that include symmetry. The results are expressed as percentage changes in the frequency of symmetric pairs relative to baseline. Scores for the metrics in the table were first computed per image and then averaged.

First note that, in the VRD-World ontology, 7 of the 71 user-defined properties (which correspond to NeSy4VRD predicates) are declared to be symmetric. These properties (predicates) are: `near`, `by`, `nextTo`, `beside`, `adjacentTo`, `acrossFrom`, and `with`.

One needs to be careful when interpreting the results in Table 4.3 because, as we saw in Section 4.4.4, combination treatments like t09 increase the number of predictions overall. Hence, there is an element of confounding at work to which we must be alert. However, treatment t01 (symmetry, alone) is relatively immune to such confounding because it does not induce the PPNN to generate more predictions per image.

We can see that, relative to baseline, treatment t01 has a strong impact on symmetric pairs. Per metric (a), the mean number of predicted visual relationships whose predicate is symmetric almost doubles (a 94% increase relative to baseline). So, treatment t01 induces the PPNN to predict more visual relationships whose predicate is symmetric. Per metric (b), the mean number of predicted visual relationships with a symmetric

predicate that occur in symmetric pairs nearly quadruples (a 286% increase relative to baseline). This is strong evidence that the PPNN is receptive to the signal sent by OWL reasoning: it is learning to predict in symmetric pairs. Further, per metric (c), the mean number of predicted visual relationships occurring in symmetric pairs that are submitted together, with the topN=50 most confident predictions, again nearly quadruples. This is strong evidence that the PPNN is learning to predict the two visual relationships in each symmetric pair with equal confidence.

Observe that treatment t14, which combines t01 and t04 (symmetry and equivalence) has an impact on symmetric pairs that exceeds that of the bigger combination treatments, t09 and t10. This is surprising. There appears to be strong affinity between the inference semantics of symmetry and that of property equivalence. This may well be due to the fact that many of the 7 symmetric properties enumerated above are also declared to be equivalent to one another. Properties `near` and `by` are declared to be equivalent. And properties `nextTo`, `beside`, and `adjacentTo` are declared to be mutually equivalent. Thus, treatment t14 provides further evidence of interesting interactions that can manifest when the sub-categories of OWL inference semantics are used simultaneously, in combinations. But we must remember the confounding at work. Treatment t04 (equivalence) strongly induces the PPNN to make more predictions overall.

4.4.9 The effects of OWL reasoning on inverse pairs

Here we examine the effects of OWL reasoning with respect the inference semantics of inverses. For treatments that include the inference semantics of inverse, we examine the frequency with which visual relationships predicted for predicates that have inverses are predicted in inverse pairs. In other words, we examine how well the subsymbolic PPNN learns which predicates have inverses, and how well it learns to act on that knowledge, by predicting those predicates in inverse pairs. Table 4.4 shows results from an analysis of the inverse pairs present (or not) in the predicted visual relationships of a selection of treatments that include inverse. Note, however, that treatment t09 does *not* declare any inverses. We include it in the table for comparison purposes. The results are expressed as percentage changes in the frequency of inverse pairs relative to baseline. Scores for the metrics in the table were first computed per image and then averaged.

First note that, in the VRD-World ontology, 10 of the 71 user-defined properties (which correspond to NeSy4VRD predicates) are declared to be involved in 5 inverse pairs. These properties (predicates) are: (above, below), (beneath, over), (over, under), (contain, inside), and (onTheLeftOf, onTheRightOf).

We can see that, relative to baseline, treatment t02 has a strong impact on inverse pairs. Per metric (a), the mean number of predicted visual relationships whose predicate has an inverse increases by 55% relative to baseline. So, treatment t02 induces the PPNN to

Table 4.4: Percentage changes in inverse pairs relative to baseline for topN=50

Metric	t02	t09	t10	t12 (t1,t2)
(a) mean predicted VRs with a predicate with an inverse	55	24	49	23
(b) mean predicted VRs appearing in inverse pairs	158	38	161	96
(c) mean predicted VR inverse pairs submitted together	154	47	158	104
(d) mean predicted VRs with a predicate with an inverse hits	1	17	5	-2

predict more visual relationships whose predicate has an inverse. We can also contrast this 55% uplift from treatment t02 with the corresponding uplift delivered by treatment t09, which does not declare any inverses at all. The uplift from t09 is just 24%, despite the fact that t09 induces the PPNN to generate many more predictions per image, on average. This is further evidence that the PPNN is sensitive to the signal that OWL reasoning sends regarding inverses in treatment t02.

Per metric (b), the mean number of predicted visual relationships whose predicate has an inverse, and that appear in inverse pairs, increases by 158% relative to baseline. This is strong evidence that the PPNN for t02 learns to predict in inverse pairs, in response to the guidance in the annotations provided by OWL reasoning. If we compare this 158% uplift from t02 with the 161% uplift from t10 (which includes t02), we see that, effectively, there is no change in the mean number inverse pairs. This appears to suggest that the inference semantics of inverses is having trouble mixing constructively with the other four sub-categories of link inference semantics present in treatment t10. If we compare the 158% uplift from t02 with the 96% uplift from t12 (which combines symmetry with inverses), we can see that in treatment t12 there appears to be a *destructive* interaction occurring between symmetry and inverses, not a constructive or neutral interaction. If so, then this destructive interaction will be playing out in treatment t10 to some extent, and this may be part of an explanation for why t10 does not induce more inverse pairs than t02.

Metric (c) in Table 4.4 is the mean number of predicted visual relationships occurring in inverse pairs that are submitted together (for the topN=50 most confidently predicted visual relationships). With respect to this metric, treatment t02 delivers an uplift of 154% relative to baseline. This is strong evidence that the PPNN is learning to predict the two visual relationship elements of an inverse pair with equal confidence. But notice how this uplift fades to a 104% uplift from treatment t12, when the inference semantics of inverses is combined with the inference semantics of symmetry. This is more evidence of interesting interactions between the sub-categories of link inference OWL reasoning, and the effects these can have on the subsymbolic learning of our PPNN.

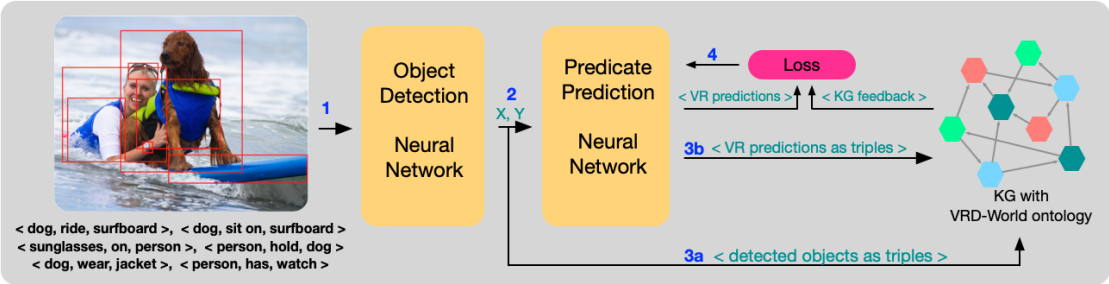


Figure 4.10: The setting for investigation 2 of thread two. The OWL-based knowledge graph hosting the VRD-World ontology, acting as a symbolic reasoning binary classifier, detects predictions emerging from the PPNN during training that are semantically invalid. A semantic loss penalty is applied to teach the predicate prediction neural network to not predict visual relationships that are semantically invalid.

4.5 Knowledge graph reasoning for applying logical constraints

In this section we describe investigation 2 of thread two of our research into combining symbolic OWL reasoning with subsymbolic learning. We start with an overview of investigation 2 that includes descriptions of relevant methods. Then we present results and discuss them.

4.5.1 Overview and methods

As we did in investigation 1, here in investigation 2 we use an OWL-based knowledge graph tool, hosting our VRD-World OWL ontology (from NeSy4VRD), and OWL reasoning, in the guise of a symbolic reasoning engine. But here in investigation 2, we use our symbolic reasoning engine for different purposes. We can think of investigation 1 as demonstrating using OWL reasoning to help teach the subsymbolic PPNN what *to* predict. The materialisation of the image scene graphs provided more positive training examples for the PPNN to learn to mimic. Here in investigation 2, we demonstrate using OWL reasoning to help teach the PPNN what *not* to predict: visual relationships that are semantically invalid. Figure 4.10 summarises the setting for investigation 2.

Figure 4.11 zooms-in on Figure 4.10 to provide more detail. It highlights the 'predicate detection' setting of our research, in which the noise of object detection is cancelled by using annotated objects in place of detected ones. It also distinguishes between the PPNN training and PPNN inference settings. The OWL-based knowledge graph, and symbolic (OWL) reasoning, behave the same in both settings, but the decisions of the real-time symbolic reasoning binary classifier lead to different actions with respect to applying logical constraints.

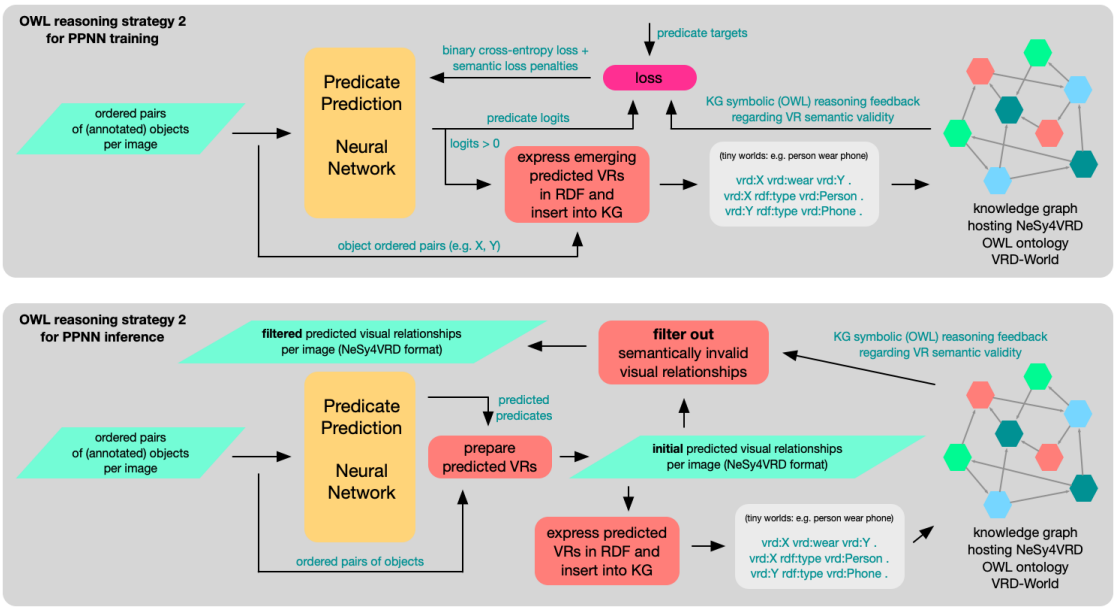


Figure 4.11: An illustration of the OWL reasoning strategy used in investigation 2, where predictions of visual relationships are validated using symbolic (OWL) reasoning. The top panel shows the setting for PPNN training, the bottom panel for PPNN inference. In both settings, an OWL-based knowledge graph hosting OWL ontology VRD-World, acting as a real-time symbolic reasoning binary classifier, validates predictions of visual relationships. During PPNN training, predictions classified as semantically invalid (*e.g.*, `dog drive surfboard`) attract semantic loss penalties, as soft logical constraints. During PPNN inference, predictions classified as semantically invalid are filtered out, as hard logical constraints.

Here in investigation 2, we use our symbolic reasoning engine to teach the PPNN the domain and range restrictions that have been declared for various of the user-defined properties (predicates) in VRD-World. These domain and range restrictions allow the OWL-based knowledge graph to differentiate between semantically valid and semantically invalid visual relationships. Thus, can say we use the symbolic reasoning engine to teach the PPNN the domain and range restrictions defined in the VRD-World ontology. The hypothesis behind investigation 2 is that, by doing so, the predictive performance of the PPNN may improve, because there will be fewer semantically invalid predictions crowding-out semantically valid ones from some topN category.

Consider a simple example. In VRD-World, `(person drive car)` is semantically valid, but `(teddyBear drive car)` is semantically invalid, because in VRD-World only people can drive cars. More specifically, `(teddyBear drive car)` is invalid because, in VRD-World, members of class TeddyBear cannot simultaneously be members of class Person. The VRD-World ontology declares that the domain of property `drive` is class `Person`. Hence, when OWL reasoning sees the triple `(teddyBear drive car)`, it infers that `(teddyBear rdf:type Person)`. If, however, `teddyBear` has separately

been declared to be a member of class (TeddyBear), with a triple such as (teddyBear rdf:type TeddyBear), then this leads to a logical contradiction. This is because, in VRD-World, class TeddyBear is a subclass of class Not_Person, and classes Person and Not_Person have been declared to be *disjoint*, meaning no individual can be a member of both classes simultaneously.

The categories of OWL reasoning exercised in investigation 2 are different from those exercised in investigation 1. In investigation 1, we leveraged 'link inference' capabilities of OWL reasoning. Here in investigation 2, we leverage 'type inference' capabilities of OWL reasoning, together with OWL's 'logical inconsistency inference' capabilities. In our case, the latter depends upon the former. As explained in the example, when individuals are inserted into our OWL-based knowledge graph, OWL infers the 'type' of that individual, *i.e.*, the class or classes of which it is a member. OWL 'type inference' may or may not give rise to a logical contradiction in VRD-World. It is OWL 'logical inconsistency inference' that looks for and detects logical contradictions. Some OWL-based knowledge graph tools—ones that do their reasoning incrementally, as triples are inserted—can catch logical contradictions in real-time, as part of the insertion transaction. We use an OWL-based knowledge graph tool such as this. If OWL reasoning detects a logical contradiction, the insertion is rejected, with the equivalent of an exception response.

Semantic loss penalties as logical constraints If, during training, the PPNN shows an inclination to predict a certain visual relationship (*i.e.*, the PPNN's confidence in a prediction rises above a threshold of 0.5), we have the OWL-based knowledge graph (our symbolic reasoning engine) evaluate it, for semantic validity. We do this by converting the PPNN's logit for the prediction into an RDF triple representation of the corresponding visual relationship, and by attempting to insert that visual relationship into the knowledge graph. If the insertion succeeds, the visual relationship is semantically valid, and we do nothing. We let the PPNN continue to learn to predict that visual relationship, if it remains inclined to do so. But if the symbolic reasoning engine rejects the insertion of the visual relationship, then we know it is semantically invalid. When such cases arise, we penalise the PPNN with a *semantic loss penalty*, to encourage it to *not* predict that particular relationship. We calculate our semantic loss penalty as a function of the PPNN's confidence in the (semantically invalid) prediction: the greater the confidence, the greater the penalty. If we let c (a probability) denote the PPNN's confidence in a hypothetical prediction, and p denote the penalty, then the equation for our semantic loss penalty is $p = -\lambda \log(1 - c)$, where the hyper-parameter λ controls the influence of all such penalties. We set $\lambda = 1$. Applying the semantic loss penalty is akin to applying a logical constraint. Once a particular relationship prediction for an image has been identified as semantically invalid, we apply our semantic loss penalty continuously, from that point onward, as long as training lasts, even after the PPNN's

confidence in the prediction falls below the threshold of 0.5. We do this in an attempt to drive the PPNN's confidence in that prediction to zero. Zero is the appropriate confidence to have for a visual relationship that is known to be semantically invalid.

A symbolic reasoning binary classifier Observe, per the discussion above, that when we attempt to insert a visual relationship into our symbolic reasoning engine, the insertion either succeeds or fails. We arranged for this to be the case, by designing our VRD-World ontology appropriately. The point here, however, is that by designing VRD-World to *enforce* all of the property domain and range restrictions declared within it, we can use our OWL-based knowledge graph (our symbolic reasoning engine) as a *symbolic reasoning binary classifier*. Thus, we can say that investigation 2 demonstrates the use of a symbolic reasoning binary classifier to help train a subsymbolic classifier (in this case, our PPNN).

Investigation 2 goes further. It uses our symbolic reasoning binary classifier in two different contexts. We use it during training, to help teach the PPNN not to predict semantically invalid relationships. And we use it during inference as well, to filter-out any semantically invalid predictions that may emerge. Note that these two use cases for the symbolic reasoning binary classifier are *independent* of one another. We could use the symbolic reasoning binary classifier during inference, to filter-out invalid predictions, with any treatment, including all those discussed in investigation 1.

Knowledge graph usage and management The symbolic reasoning computation burden associated with any given interaction with our symbolic reasoning binary classifier is small. Our use case makes it so. We use our VRD-World ontology as the counterpart of a logic program, and we only ever submit tiny worlds of facts (three related triples representing a visual relationship) for OWL to reason over, under the governance of VRD-World. Here is an example tiny world for a visual relationship:

```
(teddyBear rdf:type TeddyBear)
(car rdf:type Car)
(teddyBear drive car)
```

During PPNN training, we manage the size of the knowledge graph. Insertions for semantically invalid visual relationships fail, so the size of the knowledge graph does not change. But insertions for semantically valid visual relationships succeed, and hence the knowledge graph steadily grows in size, not just due to the three asserted triples in each interaction, but due to the many other triples that get inferred as a logical consequence. If left to grow too large, OWL reasoning would slow. So we manage the size of the knowledge graph within the training cycle. The knowledge graph starts empty except for our VRD-World ontology. Tiny worlds are put to it, in a rapid succession of interactions. We count the interactions. At the top of each training epoch, we check if an interaction

Table 4.5: Treatment definitions for OWL reasoning strategy 2 (prediction validation).

treatment name	treatment key	logical constraints applied during PPNN training	logical constraints applied during PPNN inference
t21	training only	soft (semantic loss penalties)	
t22	training & inference	soft (semantic loss penalties)	hard (filtering-out)

threshold has been breached and, if so, we tell the knowledge graph to clear itself, and then we reload VRD-World, and training resumes.

We used the freely available GraphDB OWL-based knowledge graph tool ("Ontotext GraphDB", 2023) as our symbolic reasoning engine. It behaved robustly. Integrating real-time knowledge graph interactions into a neural network training cycle of course slows the training. And, as we learned, if approached naively, the slowdown (with the freely available GraphDB, at least) can be unacceptable. But we learned to minimise the interactions with the knowledge graph to the point where the slowdown in training became reasonable, and eventually comfortable. One insight we share is: do not interact with your knowledge graph straight away. At model instantiation time, the random initialisation of the weights of the network can lead to vast numbers of *spurious* predictions emerging, which trigger vast numbers of knowledge graph interactions. Bypass these. The neural network needs time to learn for itself. We began interacting with our knowledge graph in epoch 3.

We were able to employ another tactic as well, in order to manage knowledge graph interactions. Any given visual relationship is an *instance* of a visual relationship *type*. When one instance of a type is classified (as valid or invalid, by our symbolic reasoning binary classifier), the type is classified, and hence every other instance. So we arranged to *remember* the classification decisions of our symbolic reasoning binary classifier, and to look to these first, before interacting with the knowledge graph. This enabled us to eliminate vast amounts of redundant knowledge graph interaction.T

Treatment names We use treatment 't21' to refer to neurosymbolic systems that used the symbolic reasoning binary classifier during training, to suppress semantically invalid predictions by applying the semantic loss penalty as a logical constraint. We use treatment 't22' to refer to neurosymbolic systems that used the symbolic reasoning binary classifier during training in the same way, and which also used the symbolic reasoning binary classifier during inference, to filter-out semantically invalid predictions. When we refer to treatment 't09', we refer to the same treatment 't09' discussed in Section 4.4, whose VRD-World ontology includes inference semantics for symmetry, transitivity, equivalence, and sub-properties. Table 4.5 summarises the treatments we explored in connection with the OWL reasoning strategy employed in investigation 2 (prediction validation for applying logical constraints).

Table 4.6: Comparison of semantic validity of predicted visual relationships for topN=999

Model	predicted	valid	invalid	invalidity rate
baseline	32101	32008	93	0.002904
t21	29427	29403	24	0.000795
t22	29403	29403	0	0
t09	56071	56059	12	0.000214

4.5.2 The effects of OWL reasoning on semantic validity (A)

Our baseline deep learning system (our PPNN) turned out to predict very few visual relationships that were semantically invalid. Perhaps this should not be surprising. The PPNN is trained only on semantically valid visual relationships. The fact that it tends (overwhelmingly) to make only semantically valid predictions is the sign of a good learner. But this made analysis of the effects of OWL reasoning (in the guise of our symbolic reasoning binary classifier) harder. To find evidence of the effect of OWL reasoning for the suppression of invalid predictions, we conducted a more granular analysis, at the level of the individual predicted visual relationships themselves. We analysed the effect of OWL reasoning for the suppression of invalid predictions from two perspectives: the predictions themselves, and the prediction *types*. We consider the former in this section, and the latter in the section which follows. We relied upon our symbolic reasoning binary classifier for conducting our analyses. Without it, analysis would have been infeasible. Thus, it is the case that, the symbolic reasoning binary classifier has played a role in three different contexts: training, inference, and analysis.

Table 4.6 summarises the findings for the effects of OWL reasoning on suppressing invalid predictions, from the perspective of the individual visual relationships themselves. The values in the table are mean counts: mean number of predicted visual relationships, mean number that are semantically valid, and mean number that are semantically invalid. The last column shows the rate of invalidity implied by the counts. The underlying sample size is 4.

Table 4.6 shows that treatment 't21', where we use our symbolic reasoning binary classifier to help train the PPNN by applying a semantic loss penalty to predictions of semantically invalid visual relationships, is effective. Whereas the baseline system generates 93 invalid predictions on average, treatment 't21' generates only 24. This represents a reduction of 74%. But the semantic logical constraint applied by treatment 't21', while effective, was not completely effective. One contributing factor may be that, we do not

apply the semantic logical constraint (with the help of the symbolic reasoning binary classifier) until the PPNN's confidence in a prediction rises above a threshold of 0.5. Thus, it is possible for invalid predictions to emerge late in the training cycle, shortly before early-stopping kicks-in, and before the semantic logical constraint has had an opportunity to influence learning. Another contributing factor may be the value we gave to our hyperparameter that regulates the influence of the semantic loss penalty. We assigned a value of 1. Larger values would amplify the semantic loss penalty and apply a harder constraint.

The fact that treatment 't21' is not completely successful in suppressing all invalid predictions, however, creates an opportunity to showcase the contribution that OWL reasoning can make when the symbolic reasoning binary classifier is utilised during inference, to filter out unwanted invalid predictions. Table 4.6 shows that treatment 't22', which uses the symbolic reasoning binary classifier to help train the PPNN, and to filter-out invalid predictions during inference, is completely effective. The mean number of predictions submitted for performance evaluation that are semantically invalid falls to 0.

We included treatment 't09', from investigation 1, in Table 4.6 to highlight the surprising finding that it generates fewer semantically invalid predictions than baseline, and fewer than treatment 't21' as well. This suggests that the rich 'link inference' semantics of the version of the VRD-World ontology that is used in treatment 't09' (which includes symmetry, transitivity, equivalence, and sub-properties), and the associated image scene graph materialisation delivered by these inference semantics, helps the PPNN learn to not predict semantically invalid relationships. Perhaps this effect of 't09' should not be too surprising. Treatment 't09' is trained on far larger and richer target scene graphs (of semantically valid relationships), so it gets more intensive and concentrated supervision from semantically valid relationships.

4.5.3 The effects of OWL reasoning on semantic validity (B)

Table 4.7 summarises the findings for the effects of OWL reasoning on suppressing invalid predictions, from the perspective of visual relationship *types*. The values in the table are mean counts: mean number of predicted visual relationship types, mean number that are semantically valid types, and mean number that are semantically invalid types. The last column shows the rate of invalidity implied by the counts. Again, as always, the underlying sample size is 4.

Table 4.7 shows that treatment 't21' is effective at reducing the number of semantically invalid relationship types that are predicted. Whereas the baseline system generates 55 invalid relationship types on average, treatment 't21' generates only 20. This represents

Table 4.7: Comparison of semantic validity of predicted visual relationship types for topN=999

Model	predicted	valid	invalid	invalidity rate
baseline	8651	8597	55	0.006195
t21	7693	7673	20	0.002608
t22	7693	7673	20	0.002608
t09	11979	11968	11	0.000893

a reduction of 64%. Treatment 't22' shows the same counts as 't21' due to the nature of our multi-step PPNN pipeline. Treatments 't22' processes the same predictions as 't21', but during inference, to filter-out invalid predictions (with the help of the symbolic reasoning binary classifier). If these counts were not identical, something would be wrong in our pipeline. We see that treatment 't09' does a better job of suppressing semantically invalid prediction *types* than treatment 't21' (which applies our semantic logical constraint), despite generating predictions for a far wider range of relationship types than 't21'.

When considering visual relationship *types*, and interpreting Table 4.7, it is useful to observe that the full space of structurally possible visual relationship types has size 890,624. Thus, treatment 't09', for example, generates predictions whose types span only 1.3% of the full type space available.

4.5.4 The effects of OWL reasoning on recall@N

Figure 4.12 shows the effects that treatments 't21' and 't22' have on recall@N. It shows that the effects of treatments 't21' and 't22', as measured by recall@N, are virtually identical. This result is expected, given the tiny numbers of semantically invalid predictions that exist to be filtered-out during inference by 't22'. But the recall@N scores delivered by 't21' and 't22' relative to baseline are a surprise. Given the tiny numbers of semantically invalid predications generated by the baseline system, we would have expected to see no change in recall@N for 't21' and 't22'. This is what we see for recall@25. But the 't21' and 't22' scores for the other levels of recall@N *fall* slightly, relative to baseline. This slight fall in recall@N may be explained by the lower number of predictions overall, relative to baseline, that are generated by treatments 't21' and 't22', as seen in Table 4.6. But, if so, this prompts the question of why they generate fewer predictions than baseline. It may be there is some form of side-effect from applying the semantic loss penalty that we had not anticipated and do not understand. Perhaps the influence of the semantic loss penalty is being felt more widely than expected.

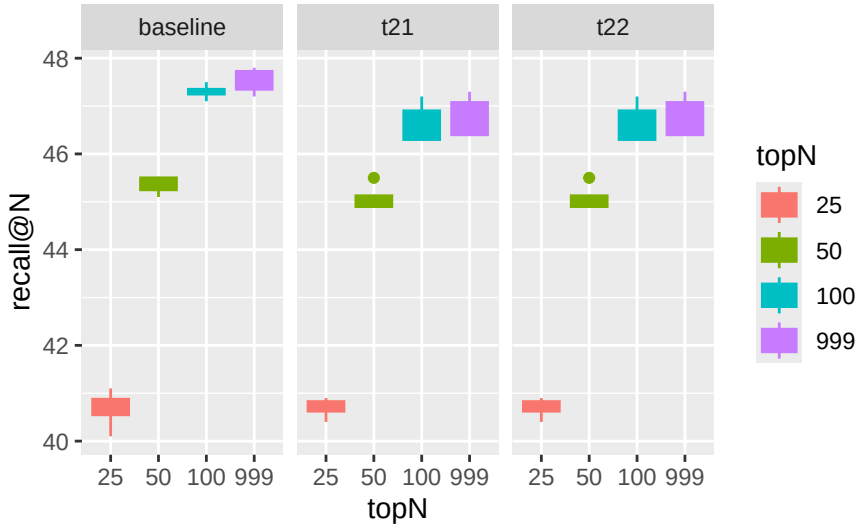


Figure 4.12: The effects of OWL reasoning, used as a logical constraint to suppress predictions of invalid visual relationships, on recall@N.

4.6 Future work

Here we describe an idea for a further investigation into ways in which OWL reasoning can be leveraged and integrated in a visual relationship detection system such as ours. We envisaged this investigation early on in our research, but have not, as yet, had the opportunity to pursue it.

4.6.1 Proactive prediction evaluation

Consider once again the OWL reasoning strategy we employed in investigation 2, where predicted VRs are symbolically evaluated to see if they are semantically valid or invalid. This represents a *reactive* use of OWL reasoning, and hence a *reactive* use of logical constraints. During PPNN training, we wait for the network's confidence in a VR prediction to rise above a probability threshold of 0.5 before evaluating it symbolically with our symbolic classifier (our OWL-based knowledge graph hosting OWL ontology VRD-World).

In contrast, one potential avenue for future work involves exploring the *proactive* use of OWL reasoning to help guide and constrain subsymbolic learning. More specifically, we think it feasible to proactively scan and evaluate the entire prediction space of our PPNN (our predicate detector), in order to pre-identify visual relationships that are *(i)* logically plausible (and, hence, should probably be predicted, because they are poten-

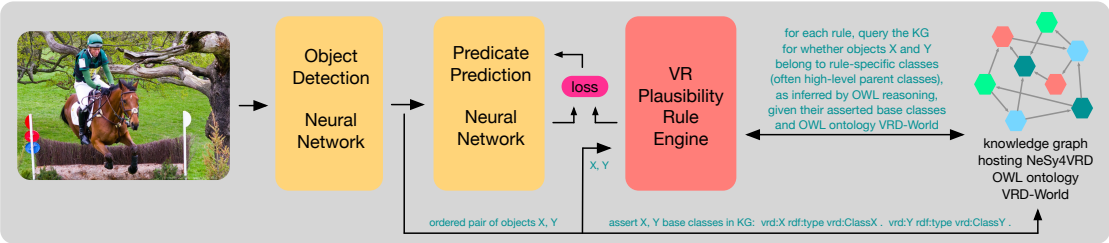


Figure 4.13: A vision of an OWL-based neurosymbolic visual relationship detection system that uses Datalog-like rules, executed in a rule engine, together with OWL-based knowledge graph reasoning and queries, to *proactively* scan the prediction space of a neural network, in order to identify logically plausible and logically implausible visual relationship predictions. Predictions of logically implausible visual relationships attract a logical constraint penalty, to discourage their prediction. Predictions that fail to emerge for logically plausible visual relationships attract a logical inducement penalty, to encourage the network to learn to predict these high-value (likely-to-be-a-hit) visual relationships.

tial hits), and *(ii)* logically implausible (and, hence, should not be predicted, because they are unlikely to be hits). This way, it should be feasible to apply semantic loss *inducements* to encourage the PPNN to learn to predict visual relationships deemed to be logically plausible, and we can apply semantic loss *penalties* to discourage the PPNN from learning to predict visual relationships deemed to be logically implausible. Figure 4.13 shows the vision for this research setting.

Datalog-like rules describing visual relationships execute in a rule engine. For example, here is a potential rule for the visual relationship pattern `(x wear y)`:

```
wear(x,y) :- WearCapableThing(x), WearableThing(y), ir(y,x) ~ 1.
```

For each ordered pair of objects (for a given image), all of the rules defined within the rule engine would be executed, in one linear pass. Just prior to execution of the rules (for a given ordered pair of objects), the classes for the two objects involved in the ordered pair, (X, Y), are asserted within an OWL-based knowledge graph hosting OWL ontology VRD-World. OWL *type* inference will then infer all of their parent classes. As each rule in the rule engine executes, the knowledge graph is queried to check if the class-specific goals of the rule are satisfied or not. For instance, for our example `wear(x,y)` rule, the knowledge graph would be queried to see if object `x` had been inferred to be a `WearCapableThing`, or not. If all of the goals for a rule are satisfied, that visual relationship (for that specific ordered pair of objects) would be deemed logically plausible. Otherwise, it would be deemed logically implausible. Note that, in the example `wear(x,y)` rule, the third goal is an *inclusion ratio* condition on the bounding boxes of the two objects. This represents an extension to OWL's logical inference capabilities, and it serves to illustrate why one might wish to employ Datalog rules and a rule engine in combination with OWL-based knowledge graph reasoning.

For the semantic loss *penalties* to which we refer, the basic idea is: the *greater* the network's confidence in an implausible (unwanted) VR prediction, the greater the penalty. This way, predictions of implausible (unwanted) VR predictions can be suppressed. One can imagine using the very same formulation for a semantic loss penalty that we used in investigation 2: $penalty = -\lambda_1 \log(1 - c)$, where c (a probability) denotes the PPNN's confidence in a hypothetical prediction, and where the hyper-parameter λ_1 controls the influence of the penalties. However, it may well prove desirable to explore other formulations as well, to see which is most effective.

For the semantic loss *inducements* to which we refer, the basic idea is: the *lower* the network's confidence in a plausible (likely-to-be-a-hit) VR prediction, the greater the inducement. This way, predictions of plausible (high-value) VR predictions can be induced or encouraged. One candidate formulation for a semantic loss inducement is a simple variation of the semantic loss penalty formulation: $inducement = -\lambda_2 \log(c)$, where the hyper-parameter λ_2 controls the influence of the inducements.

4.7 Summary

This chapter presented thread two of our research into combining subsymbolic learning with symbolic OWL reasoning in neurosymbolic systems. Thread two uses the computer vision application task of detecting visual relationships in images as the context within which to study our subject. We opened the chapter by defining the problem of detecting visual relationships in images, in order to set the scene for what was to come. Then we presented our baseline deep learning visual relationship detection system. We described its key components (an object detector, and a multi-class, multi-label classifier), and we explained how we worked with them for conducting our two investigations.

Then we discussed the first of our two investigations. Investigation 1 involved leveraging OWL reasoning in order to enrich the supervision received by our PPNN (our predicate prediction neural network) during its training. We explained how we leveraged five different sub-categories of OWL 'link inference' capability, individually, and in combinations, to materialise the NeSy4VRD annotated scene graphs of our images, thereby augmenting the scene graphs in a variety of different ways. We saw that our PPNN was surprisingly sensitive to, and responsive to, the variations in the augmentations of the image scene graphs, and to the corresponding variations in the training supervision engendered by those variations in the augmentations of the image scene graphs. We examined the responses of our PPNN to the OWL-driven variations in the training supervision it received from a variety of angles. We looked for the effects of OWL link inference on predictive performance, as measured by recall@N, and saw rich variety in the changes to recall@N that emerged in response to different mixtures of

OWL link inference during scene graph augmentation. We saw that certain mixtures of OWL link inference led to highly significant increases in recall@N, and that others led to small decreases. We looked for the effects of OWL link inference on the volume of visual relationship predictions generated by our PPNN during inference, and saw how one particular mixture of OWL link inference was able to induce the PPNN to generate far more predictions per image, on average, than our baseline deep learning system was able to do. We examined the effects of OWL link inference on mAP@N, and found it to be remarkably robust given the (often) corresponding increases in the volume of predictions induced by the link inference. We examined the effects of OWL link inference on mean hits per image, and saw that the PPNN responses here mirrored those observed for recall@N. We saw that, as a result of OWL link inference, the speed with which our PPNN learned tended to increase, sometimes dramatically. We examined the effects of the inference semantics of symmetry, and saw that when inference for symmetry is involved in OWL's augmentation of image scene graphs, our PPNN better learns to predict symmetric pairs. Similarly, we examined the effects of the inference semantics of inverses, and saw that when inference for inverses is involved in OWL's augmentation of image scene graphs, our PPNN sometimes better learns to predict inverse pairs, and sometimes not.

Investigation 2 involved leveraging OWL reasoning in order to teach our PPNN to not predict visual relationships that are semantically invalid (according to our VRD-World ontology). We explained how our strategy for doing this involved applying a semantic loss penalty during training, in a manner akin to applying a logical constraint, whenever the PPNN showed an inclination to predict something semantically invalid. We saw that the key to this strategy was our VRD-World ontology, that we designed to enable OWL reasoning to enforce all of the domain and range restrictions declared within it. We explained how we used our VRD-World ontology as a counterpart of a logic program, in a real-time symbolic reasoning engine (an OWL-based knowledge graph tool), and how, during PPNN training, we presented our engine with only tiny worlds over which to reason at any one time. And we explained how the net effect of all this implied that our OWL-based knowledge graph, hosting our VRD-World ontology, played the role of a symbolic reasoning binary classifier. We also described how we were able to leverage our symbolic reasoning binary classifier during PPNN inference, to identify semantically invalid visual relationship predictions so they could be filtered-out from the predictions submitted for performance evaluation. Having laid this groundwork, we then examined the effects of using our symbolic reasoning binary classifier to enable us to apply semantic loss penalties to help guide PPNN learning. We looked at how our strategy effected the frequency of semantically invalid predictions. We saw that then used during training, our strategy reduced the frequency of semantically invalid predictions dramatically, but not completely. And we saw the when we combined our training strategy with use of the symbolic reasoning binary classifier during inference,

to filter-out invalid predictions, the frequency of invalid predictions fell to 0. We looked at how our strategy effected the number of semantically invalid visual relationship *types* that our PPNN predicted, and we saw that here again, our strategy reduced the number of *types* of invalid visual relationships significantly. We also observed how our strategy appeared to reduce the number of predictions generated by our PPNN, and that this had a knock-on effect of delivering slightly lower recall@N scores.

We closed by describing an idea for future 'thread two' work that involves extending OWL reasoning with Datalog-like rules. We described our vision of a simple rule engine that executes common-sense rules, in co-operation with an OWL-based knowledge graph acting as a symbolic reasoning engine, to decide on plausible visual relationship predictions for a given ordered pair of objects. And we explained how these suggestions of plausible predictions might be leveraged to apply *semantic plausibility penalties* that encourage the PPNN to predict these plausible visual relationships if it is not already doing so.

Finally, recall our central research questions: *(i)* how can we combine subsymbolic learning with symbolic OWL reasoning, and *(ii)* what are the effects or benefits of doing so? This chapter (thread two of our research) addressed these questions from the perspective of the task of visual relationship detection in images. Thread two of our research demonstrates novel use cases for Semantic Web, OWL-based knowledge graph technologies generally. It shows how a custom-designed OWL ontology can be used as a counterpart of a logic program, and how an OWL-based knowledge graph tool hosting that ontology can be used as a symbolic reasoning engine in neurosymbolic systems. It demonstrates the use of such a symbolic reasoning engine in batch-mode and in real-time, and during network model training, network model inference, and during analysis of experiment results (by enabling analyses that would otherwise have been infeasible). It shows that different aspects of OWL reasoning capability (link inference, type inference, and logical consistency inference) can be leveraged discretely to facilitate specialised applications in neurosymbolic systems. Overall, thread two demonstrates a variety of ways in which symbolic OWL reasoning can be effectively combined with neural, subsymbolic learning in neurosymbolic systems. It shows that Semantic Web technologies, OWL ontologies, and OWL-based knowledge graph tools have a place in neurosymbolic systems research.

Chapter 5

Tensor Knowledge Graphs

Recall that our research into combining subsymbolic learning with symbolic OWL reasoning in neurosymbolic systems has three threads. Chapter 3 covered thread one of our research: our NeSy4VRD resource for facilitating neurosymbolic research with OWL-based knowledge graphs in the computer vision task of detecting visual relationships in images. Chapter 4 covered thread two of our research: using OWL-based knowledge graphs and symbolic OWL reasoning to guide neural, subsymbolic learning for the task of visual relationship detection in images. This chapter covers thread three of our research: tensor knowledge graphs. We define the concepts of an OWL-based tensor knowledge graph, and of tensor knowledge graph reasoning. And we consider applications of these concepts in neurosymbolic systems.

We open this chapter by explaining our notion of an OWL-based tensor knowledge graph. Then we present our specification of a binary tensor representation for an OWL-based knowledge graph, point out its limitations, and show evidence that it works. Next we provide a thorough overview of our tensor knowledge graph reasoning techniques for emulating aspects of symbolic OWL reasoning. Included within that overview, we do the following:

- we introduce key relational operations that we employ as re-useable reasoning building blocks;
- we show how tensor knowledge graph reasoning is grounded in relational mathematics by illustrating how we use the relational operations to emulate the inference semantics of OWL constructs `owl:inverseOf` and `rdfs:domain`
- we summarise the RDFS and OWL inference semantics supported by the current incarnation of our tensor knowledge graph reasoning engine;
- we dive deep into two matrix-based transitive closure algorithms, because the

efficiency of our transitivity inference has an important influence on the overall practical viability of our tensor knowledge graph reasoning engine;

- we describe early-stopping conditions that we have identified for use with one of the two transitive closure algorithms that we work with, and we show that these stopping conditions make that algorithm significantly faster than it would otherwise be, and without compromising the logical soundness of the inference;
- and we compare the runtime efficiency of our two transitive closure algorithms, and show that, counter-intuitively, the algorithm with greater computational complexity is actually much faster than the algorithm with lower complexity.

Once the concepts of tensor knowledge graph and tensor knowledge graph reasoning have been firmly established, we shift the focus to considering applications of these concepts, especially within neurosymbolic systems. We introduce a novel subsymbolic-symbolic neural network architecture and show how it permits tensor knowledge graph-encoded symbolic knowledge to be injected into the symbolic component of this architecture, to permit predictions of base classes to be generalised to their parent classes, according to a class hierarchy defined in an OWL ontology. We discuss a strategy for having the network learn the encoded symbolic knowledge of a class hierarchy rather than simply injecting it. We introduce the notion of 'neural symbolic inference', and demonstrate it by fitting the symbolic component of a subsymbolic-symbolic network with a tensor knowledge graph 'transitive closure' inference algorithm suitably adapted to the context of a neural network. Then we show that this adapted subsymbolic-symbolic network can successfully (and symbolically) infer the complete transitive closure of the class hierarchy of the NeSy4VRD OWL ontology, VRD-World—all within the rhythm of a conventional neural network training cycle.

Next we discuss several categories of planned and potential future work in relation to tensor knowledge graphs. Finally, we close the chapter (thread three of our research) with a summary.

5.1 The idea of an OWL-based tensor knowledge graph

We begin by reminding ourselves of the definitions of two different types of binary relation. An *homogenous* binary relation on a set X is defined to be a subset of the Cartesian product $X \times X$, the set of all ordered pairs (a, b) where a and b are both elements of X. A *heterogeneous* binary relation on two sets X and Y is defined to be a subset of the Cartesian product $X \times Y$, the set of all ordered pairs (a, b) where a is in X and b is in Y (Givant, 2017; Schmidt & Ströhlein, 1993). In other words, a binary

relation (of either category) is a set of ordered pairs.

There are two commonly used alternate representations for binary relations: 1) a directed graph, and 2) a binary matrix (Schmidt & Ströhlein, 1993). Description logics and the Semantic Web (including OWL) use alternate representation (1), a choice which leads directly to the use of `(s,p,o)` *triples*, the discrete elements of directed graphs. For our (binary) tensor representation of OWL-based knowledge graphs, we focus on alternate representation (2): binary matrices. Our tensor knowledge graph reasoning techniques rely primarily on the binary matrix representation too, although there are stages where we revert to the 'set of ordered pairs' representation of a binary relation instead.

Roughly speaking, an OWL ontology (or OWL-based knowledge graph) can be regarded as a collection of binary relations, expressed as sets of `(s,p,o)` *triples*, where the triples that belong to a given binary relation all share the same predicate, `p`. For example, to declare an OWL class hierarchy, a hypothetical OWL ontology contains an homogenous binary relation for property `rdfs:subClassOf` that is defined on the set of classes declared in the ontology. That is, the OWL ontology contains a set of triples `(s, rdfs:subClassOf, o)`, where `s` and `o` are ontology classes. Similarly, to declare that certain properties in an OWL ontology may have domains that are restricted in some way, an OWL ontology contains a heterogeneous binary relation for property `rdfs:domain` that is defined on set of user-defined properties and on the set of classes. That is, the OWL ontology contains a set of triples `(s, rdfs:domain, o)`, where `s` is a user-defined property and `o` is a class.

Our idea for representing OWL-based knowledge graphs as binary tensors is to represent each binary relation present in an OWL ontology (or OWL-based knowledge graph), whether homogenous or heterogeneous, as a binary matrix. The binary matrix representations of the OWL ontology binary relations that are of like dimension are grouped into 3D binary tensors such that the binary matrix encoding of a given binary relation occupies a certain *channel* in a 3D volume.

It is important to see that our tensor representation of an OWL-based knowledge graph seeks simply to alter the representation of the binary relations composing the knowledge graph. Our conversion from the directed graph (triples) representation of the binary relations, to a binary matrix representation of the binary relations, is just that: a switch of representation only. Within the current limitations of our work, there is no loss of information associated with this conversion of representation. It is also important to see that our tensor representation of an OWL-based knowledge graph remains entirely symbolic. We use tensors, but they are not meant to connote anything subsymbolic.

5.2 Our specification for an OWL-based tensor knowledge graph

Here we specify our (binary) tensor representation of a generic OWL-based knowledge graph. For convenience, we often refer to an instance of this representation as a *tensor knowledge graph*. We enumerate and specify the components of the representation, we discuss its current limitations, and then we show evidence (with respect to our mini VRD-World ontology) that the representation is accurate.

5.2.1 Our specification

Three ordered lists of names The first component of our tensor representation of an OWL-based knowledge graph (or OWL ontology) is a collection of lists of the names of things appearing in the OWL-based knowledge graph (or OWL ontology). To construct a tensor knowledge graph for a given OWL ontology (knowledge graph), we first extract three categories of names of things: class names, user-defined property names, and names of individuals. We sort each list of names alphabetically and preserve them. Strictly speaking, these lists of names are essential components of an overall tensor knowledge graph because we rely upon them entirely to both construct a tensor knowledge graph instance and to deconstruct it (*i.e.*, convert the binary relations encoded within its binary matrices back to their original OWL directed graph triple representations). This is because, at all times, the positions of the names in these lists (their indices) map directly to fixed rows and/or columns (with the same indices) in the binary matrices we use to encode the binary relations of the ontology. But beyond these input/output functions (which have to do with representation conversion only), these lists of names play no role whatsoever. They are not relevant to tensor knowledge graph reasoning. Hence, we consider these lists of names to be a sort of pseudo-component of a tensor knowledge graph, and we largely disregard them.

Tensor A: homogenous relations on classes Some binary relations appearing in an OWL ontology are homogenous relations defined on the set of ontology classes. Our tensor representation of an OWL-based knowledge graph currently supports two of these binary relations: the `rdfs:subClassOf` relation and the `owl:equivalentClass` relation. We encode each of these relations in a $C \times C$ binary matrix, where C is the number of classes. We use the sorted list of class names to put 1s in the correct cells. The two binary matrices occupy the channels of a $2 \times C \times C$ tensor we call *Tensor A*.

Tensor B: homogenous relations on user-defined properties Some binary relations appearing in an OWL ontology are homogenous relations defined on the set of

user-defined object properties. Our tensor representation of an OWL-based knowledge graph currently supports three of these binary relations: the `rdfs:subPropertyOf` relation, the `owl:equivalentProperty` relation, and the `owl:inverseOf` relation. We encode each of these relations in a $P \times P$ binary matrix, where P is the number of user-defined properties. These three binary matrices occupy the channels of a $3 \times P \times P$ tensor we call *Tensor B*.

Tensor C: homogenous relations on individuals Many binary relations appearing in an OWL ontology (or knowledge graph) are homogenous relations defined on the set of individuals that appear in data triples. These relations correspond to user-defined object properties. Their names are user-defined, and there can be any number of them. Our tensor representation of an OWL-based knowledge graph currently supports any number of user-defined object properties. We encode each of these relations in an $N \times N$ binary matrix, where N is the number of individuals. These binary matrices occupy the channels of a $P \times N \times N$ tensor we call *Tensor C*.

Tensor D: heterogeneous relations on individuals and classes Some binary relations appearing in an OWL ontology are heterogeneous relations defined on the set of individuals and on the set of classes. Our tensor representation of an OWL-based knowledge graph currently supports the single most important such binary relation: the `rdf:type` relation, that maps individuals to classes. We encode this relation in an $N \times C$ binary matrix, where N is the number of individuals and C the number of classes. This binary matrix occupies the single channel of a $1 \times N \times C$ tensor we call *Tensor D*.

Tensor E: heterogeneous relations on properties and classes Some binary relations appearing in an OWL ontology are heterogeneous relations defined on the set of user-defined properties and on the set of classes. Our tensor representation of an OWL-based knowledge graph currently supports two of these binary relations: the `rdfs:domain` relation and the `rdfs:range` relation. We encode each relation in a $P \times C$ binary matrix, where P is the number of user-defined properties and C the number of classes. These binary matrices occupy the channels of a $2 \times P \times C$ tensor we call *Tensor E*.

Handling user-defined property characteristics The RDF property `rdf:type` discussed in relation to Tensor D is also used to relate user-defined properties to certain pre-defined, special-purpose OWL classes. For example, in VRD-World, the user-defined object property (binary relation) `beside` is declared to be symmetric using the triple `(vrd:beside rdf:type owl:SymmetricProperty)`. That is, stating that property `beside` is a member of class `owl:SymmetricProperty` is the OWL way of

declaring that property (binary relation) `beside` has the characteristic of being symmetric.

In OWL, user-defined properties (binary relations) can be declared to have one or more of the following characteristics: symmetric, asymmetric, transitive, functional, inverse functional, reflexive, irreflexive. Our tensor representation of an OWL-based knowledge graph extracts and records all of the characteristics defined for each user-defined object property. Unlike the lists of names discussed above, however, these object property characteristics are critical to tensor knowledge graph reasoning, just as they are to conventional OWL reasoning. They are thus an essential element of a tensor knowledge graph, but for now the characteristics of the user-defined properties are stored in a Python dictionary rather than being represented in matrices in some fashion.

5.2.2 Current limitations

At this stage in its development, our tensor representation for OWL-based knowledge graphs is rather basic relative to the full expressiveness of OWL. There are currently several significant limitations to what our tensor knowledge graph attempts to represent relative to a generic OWL ontology. We discuss each of the key limitations, in turn.

Simple class descriptions only One limitation is that our tensor knowledge graph currently handles only the most simple of OWL class descriptions. It can handle classes that are so simple they have no description, just a name. And it can handle classes with the simplest descriptions—descriptions that simply name another single class, such as with triples like `(classA rdfs:subClassOf classB` and `(classC owl:equivalentClass classD`. But it does not currently handle OWL anonymous classes (aka blank nodes) or complex class descriptions that, for example, are expressed as intersections of multiple other classes, or as unions of multiple other classes. Neither does it handle classes described as the set of individuals possessing a certain property or mix or properties. Interpreting arbitrarily complex OWL class descriptions and representing these descriptions in a way conducive to logical inference techniques based on matrix operations on binary matrices is a non-trivial problem that is left for future work. We do have insights and ideas with which to pursue this future work, however.

No data properties or literals Another key limitation is that our tensor knowledge graph currently does not handle OWL data properties, which means it does not handle literals either (*i.e.*, the literal values of data properties, like numbers, strings, dates, etc..). This limitation, however, is currently of little consequence with respect to being able to emulate OWL reasoning because the main way that data properties and literals can participate in OWL reasoning is by being mentioned in complex class

descriptions, which, as described, are not currently supported. Further, we are in good company in not supporting data properties and literals. No knowledge graph embedding models that we are aware of currently support literals either, and hence their support for data properties is limited or non-existent as well.

5.2.3 Evidence that our tensor representation is accurate

To confirm that our tensor knowledge graph faithfully represents the binary relations of our mini VRD-World ontology, we use an ever-expanding test script to show us visually what has been represented internally. Using this approach, we have evaluated every OWL binary relation currently supported by the tensor knowledge graph and confirmed that its internal representation of the OWL binary relations of mini VRD-World are correct. As evidence of this, we show output that our test script writes to the console of our integrated development environment for two selected OWL binary relations. Observe that the output written to the console for a given OWL binary relation is simply the 'set of ordered pairs' representation of the tensor knowledge graph's internal binary matrix representation of that binary relation.

For example, in mini VRD-World, the binary relation `owl:equivalentClass` consists of the following two triples:

```
vrd:Bicycle owl:equivalentClass vrd:Bike .
vrd:Brolly owl:equivalentClass vrd:Umbrella .
```

Listing 5.1 shows that if we query the tensor knowledge graph for its internal representation of relation `owl:equivalentClass`, we can confirm that its internal representation is correct.

Listing 5.1: Tensor KG representation of relation owl:equivalentClass.

```
testing owl:equivalentClass

asserted owl:equivalentClass axioms
[3 4] Bicycle Bike
[ 5 36] Brolly Umbrella
triple count: 2
```

Similarly, Listing 5.2 allows us to confirm that the tensor knowledge graph's internal representation of OWL binary relation `owl:inverseOf` is correct.

Listing 5.2: Tensor KG representation of relation owl:inverseOf.

```
testing owl:inverseOf

asserted owl:inverseOf axioms
```

```
[0 1] above below
[ 2 15] beneath over
[15 18] over under
triple count: 3
```

For now, we use our mini VRD-World OWL ontology to drive all of our research and development associated with tensor knowledge graphs. A listing of this small ontology is provided in Appendix C. The introductory annotations documented within the ontology provide comments to aid interpretation and are worth reading.

5.3 Tensor knowledge graph reasoning

Here we describe the techniques we have developed for manipulating the binary relations encoded within an OWL-based tensor knowledge graph in order to perform logical inference that emulates many aspects of OWL reasoning (within declared limitations). As will be seen, these techniques rely heavily upon familiar matrix operations like transposition, addition, multiplication, and clamping. But we also employ novel manipulations as well. And, at times, we switch from the binary matrix representation of binary relations to the 'set of ordered pairs' representation so that we can process the elements of a binary relation individually.

The idea behind tensor knowledge graph reasoning is this: perform logical inference when the binary relations in question are represented as binary matrices rather than as directed graphs of `(s,p,o)` text triples. To drive our development of these techniques, we used our mini VRD-World ontology as our guide. For each binary relation within mini VRD-World, the triples that OWL reasoning infers were the targets for our reasoning techniques to emulate.

First we give examples of the reasoning techniques we have developed to handle a selection of specific OWL ontology binary relations. Then we discuss the current limitations of our tensor knowledge graph reasoning techniques. Next we specify precisely which RDFS and OWL entailment rules are currently supported by our tensor knowledge graph reasoning techniques. Following this, we discuss details of the two (known) matrix-based transitive closure algorithms that we have adapted for use within our tensor knowledge graph reasoning techniques, one of which we call the *union-of-powers* algorithm. Then we describe the analysis we conducted to understand the behaviour of this union-of-powers algorithm, and how this analysis enabled us to identify early-stopping conditions that allow us to stop the algorithm far earlier than the worst-case, upper-bound number of iterations nominally required.

5.3.1 Key relational operations

Here we describe three operations on binary relations that are defined in the theory of binary relations that goes by the (perhaps unfamiliar) name of the *Calculus of Relations* (*e.g.*, Tarski (1941)). We provide background to the Calculus of Relations in Section 2.5. For each of the three operations, first we define it, and then we explain our implementation. We use these generic (binary) relational operations repeatedly within our tensor knowledge graph reasoning techniques. Indeed, they are cornerstones of tensor knowledge graph reasoning.

Relational union If A and B are relations on a set U, then the *union* of A and B is the relation consisting of the pairs that are either in A or in B,

$$A \cup B = \{(\alpha, \beta) : (\alpha, \beta) \in A \text{ or } (\alpha, \beta) \in B\}.$$

We implement the relational *union* operation using binary matrix addition, followed by clamping the result matrix (*i.e.*, all its elements) to a maximum value of 1. The addition of two binary matrices will, of course, sometimes yield values of 2. We can think of these as representing duplicates. The clamping suppresses any duplicates and ensures a Boolean result is delivered by our relational *union* operation. We think of this implementation for the relational *union* operation as Boolean matrix addition. The idea of using Boolean matrix addition for the relational *union* operation is not heuristic; it is grounded in the Calculus of Relations. Our approach of using standard matrix addition followed by clamping is just our way of doing Boolean matrix addition.

Relational composition Composing two binary relations A and B is like composing two functions: the composition consists of the pairs (α, β) such that A maps α to some γ, and B maps γ to β,

$$A \mid B = \{(\alpha, \beta) : (\alpha, \gamma) \in A \text{ and } (\gamma, \beta) \in B, \text{ for some } \gamma \in U\}.$$

We implement the relational *composition* operation using matrix multiplication (followed by clamping at 1). The idea of using matrix multiplication for relational composition is not heuristic; it is grounded in the Calculus of Relations. When binary relations are represented as binary matrices, Boolean matrix multiplication is the natural way to approach relational composition. The only thing unique to our approach, perhaps, is the use of clamping to ensure we always get Boolean results from standard matrix multiplication. If we let A and B represent 3-by-3 binary matrices encoding binary relations, and we let $\odot$ denote relation composition, then what we achieve with our implementation is this:

$$A \odot B = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Note that in Description logics (*e.g.*, Baader et al., 2017), the concept of relational composition is called *role composition*. In OWL, relational composition manifests most prominently in connection with *property chains* and OWL's `owl:propertyChainAxiom` property. Property chains are the OWL counterpart of Description logic role composition.

It is helpful to see that relational composition can be viewed as a generalisation of transitivity. With transitivity, one relation is composed with itself. With relational composition, different relations can be composed. For example, if R is a transitive binary relation, we have: if aRb and bRc, then aRc. But with relation composition, we can have: if aR_1b and bR_2c, then aR_3c. As we will see when we discuss matrix-based transitivity closure algorithms, this close connection between relational composition and transitivity means relational composition can be used to find transitive closures.

Relational converse Forming the *converse* of a binary relation A is a *unary* operation. The *converse* of relation A has all the pairs of A, but the elements of each pair swap positions, such that if A contains the ordered pair (α, β), then the converse contains the ordered pair (β, α),

$$A^{-1} = \{(\alpha, \beta) : (\beta, \alpha) \in A\}.$$

We implement the relational *converse* operation with matrix transposition. This is not heuristic; it is grounded in the Calculus of Relations. It is also intuitively obvious. If we let A be a 3-by-3 binary matrix encoding a binary relation, and we let superscript T denote transposition, then what we achieve with our implementation is this:

$$A^T = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^T = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

5.3.2 Emulating owl:inverseOf inference semantics

The inference semantics associated with OWL property `owl:inverseOf` are (largely) captured in the following rule (whose expression we have tailored for our forthcoming example):

```
if (above owl:inverseOf below) and (sky above car),
then (car below sky).
```

We say 'largely captured' because property `owl:inverseOf` is bi-directional (*i.e.*, symmetric). Hence `(above owl:inverseOf below)` implies `(below owl:inverseOf above)`. We must attend to this inference as well. We step through a concrete example of how our tensor knowledge graph reasoning techniques allow us to emulate all of these inference semantics associated with OWL's property `owl:inverseOf`.

Ontology and data One axiom in our mini VRD-World OWL ontology is `(above owl:inverseOf below)`, which declares that our user-defined object property `above` is the inverse of user-defined property `below`. Our mini VRD-World ontology comes with accompanying data triples, where hypothetical individuals are related to one another using the user-defined object properties of the ontology. The data triples accompanying our mini VRD-World ontology that use the user-defined object properties `above` or `below` are these:

```
     hat100 above shoes100
umbrella100 above person100
     sky101 above car101
  person100 below umbrella100
```

Tensor knowledge graph instantiation During instantiation of the tensor knowledge graph for our mini VRD-World ontology, the axiom `(above owl:inverseOf below)` is encoded within the $P \times P$ binary matrix used to represent the binary relation `owl:inverseOf` (where P is the number of user-defined object properties). This matrix occupies a particular channel of the $3 \times P \times P$ Tensor B of our tensor knowledge graph. The data triples using property `above` are encoded in an $N \times N$ binary matrix occupying a particular channel of the $P \times N \times N$ Tensor C of our tensor knowledge graph (where N is the number of individuals). The triples using property `below` (just one, in this case) are similarly encoded, and the binary matrix for relation `below` occupies some other channel of Tensor C.

For example, suppose, for simplicity, that our mini VRD-World ontology contains just $P = 4$ user-defined property names: [`above`, `below`, `over`, `under`]. Further suppose the following example `owl:inverseOf` relation consisting of two axioms (*i.e.*, a set of two ordered pairs):

```
above owl:inverseOf below
 over owl:inverseOf under .
```

Further assume that the same (ordered) list of property names is used to construct the 4×4 binary matrix that encodes this example `owl:inverseOf` relation. This matrix, denoted A, is given by:

$$A = \begin{pmatrix} 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Axiom inference When reasoning is activated, the first thing we do in relation to emulating the inference semantics of OWL property `owl:inverseOf` is to do what we

call *axiom inference*. We start with axiom inference for property `owl:inverseOf` because this OWL property is bi-directional, *i.e.*, the binary relation `owl:inverseOf` has the characteristic of being symmetric. That is, `(above owl:inverseOf below)` implies `(below owl:inverseOf above)`. Our goal with this axiom inference is to infer (materialise) all of the `owl:inverseOf` axioms that are implied (entailed) by the ones declared, given that OWL property `owl:inverseOf` is symmetric. We implement logical inference for the relation characteristic of *symmetry* using our implementation for the binary relation operation *converse*, followed by our implementation for the binary relation operation *union*. More precisely, if let A denote the binary matrix that encodes the binary relation `owl:inverseOf`, then the logical inference we perform can be described most generally as

$$A^T \cup A.$$

In other words, we first take the transpose of A (to get the converse relation, A^T), then we add the binary matrix for the converse relation, A^T, to the binary matrix for relation A, and then we clamp A at 1. The outcome of this operation is that matrix A now encodes both the asserted OWL axiom `(above owl:inverseOf below)`, and its inferred counterpart, axiom `(below owl:inverseOf above)`. Observe that this logical inference solution for *symmetry* is *not* specific to OWL property `owl:inverseOf` in any way. It is our solution for handling symmetry, in general. OWL property `owl:inverseOf` just happens to be symmetric. This general solution for handling symmetry is used repeatedly throughout our tensor knowledge graph reasoning engine—everywhere that symmetry comes into play, which is a lot.

For a concrete example, let A again denote the 4×4 matrix encoding our example `owl:inverseOf` relation, as defined above. We can infer the *symmetric closure* of this symmetric binary relation (and of every other) using the procedure specified:

$$\begin{aligned} A^T \cup A &= \begin{pmatrix} 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{pmatrix}^T \cup \begin{pmatrix} 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \end{pmatrix} \cup \begin{pmatrix} 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \mathbf{1} & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 0 \end{pmatrix}. \end{aligned}$$

The result matrix (the symmetric closure relation) now encodes 4 `owl:inverseOf` axioms, which we render here as ordered pairs: (above, below), (below, above), (over,

under), (under, over). That is, two new axioms have been inferred. They were entailed, but implicit, and have now been inferred and materialised. Our reasoning procedure, grounded in relational mathematics, has inferred new knowledge—knowledge identical to what conventional OWL reasoning would infer given the same scenario.

Data inference At this point, we know we have access to all `owl:inverseOf` axioms, because we first took care to materialise any that were implicit. Let A continue to denote the binary matrix encoding the (now materialised) axioms of binary relation `owl:inverseOf`. We can now turn to emulating the OWL reasoning associated with the inference semantics of property `owl:inverseOf` as it pertains to data triples that reference properties that have inverses, like our example triples listed earlier that use properties `above` and `below`. The first thing we do for `owl:inverseOf` data inference is to convert the binary matrix A representation of (now materialised) binary relation `owl:inverseOf` to its 'set of ordered pairs' representation. We do this because we now want to process each distinct `owl:inverseOf` axiom individually. To perform this conversion, we get the nonzero elements of A, which results in a set of ordered pairs, where the coordinates of the pairs are row and column indices per the $P \times P$ matrix A. Then we iterate over this set of ordered pairs (this set of axioms), and process each one, in turn.

To describe our `owl:inverseOf` data inference solution, we consider a single ordered pair (a single axiom). Let $[i, j]$ denote the ordered pair representation for axiom (`above owl:inverseOf below`), where, in this case, i and j are the integer indices for the user-defined properties `above` and `below`. The binary matrices encoding the relations `above` and `below` are in particular channels of the $P \times N \times N$ Tensor C of our tensor knowledge graph. Let R and S denote these two $N \times N$ matrices, respectively. We retrieve these two matrices using indices i and j. Then we get the *converse* of the relation encoded in R (for property `above`) and *union* that with the relation encoded in S (for property `below`). More precisely, to emulate the logical inference associated with OWL property `owl:inverseOf` as it pertains to data triples, we do the following:

$$R^T \cup S.$$

The resulting matrix is an updated `below` relation, an updated matrix S, that now encodes two additional (inferred) triples:

```
shoes100 below hat100
  car101 below sky101 .
```

Note that the converse of R, matrix R^T, will contain a 1 that corresponds to an inference of the triple `person100 below umbrella100`. But, as we saw at the top of our example, this triple is asserted in the data accompanying our mini VRD-World ontology, and

so that triple is already encoded matrix S. So when the *union* operation takes place between R^T and S, our matrix addition approach to relation *union* will generate a value of 2 for the cell that corresponds to triple `person100 below umbrella100`. The asserted triple, and the inferred triple, are duplicate facts. But our clamping at 1 suppresses the duplication, just as would conventional OWL reasoning (but by other means).

To make things more concrete, we now illustrate the same scenario, for `owl:inverseOf` ordered pair (above, below), using actual matrices. Recall that we are imagining the following set of data triples that use either property 'above' or 'below':

```
    hat100 above shoes100
umbrella100 above person100
    sky101 above car101
 person100 below umbrella100 .
```

If we also assume that the set of objects mentioned in these four triples constitutes the full set of objects, then we have $N = 6$ unique objects, and the binary matrices encoding these two property relations will be 6×6. Further assume that the ordered list of names for these 6 objects is as follows: [`car101`, `hat100`, `person100`, `shoes100`, `sky101`, `umbrella100`]. If let R denote the matrix encoding binary relation 'above', and S denote the matrix encoding binary relation 'below', then, given the conditions specified, these two matrices will been instantiated as follows:

$$R = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \end{pmatrix} \qquad S = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Now, to infer and materialise new ordered pairs for relation 'below' (represented by

matrix S), we invoke the procedure specified, as follows:

$$
\begin{aligned}
R^T \cup S &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \end{pmatrix}^T \cup \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \cup \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.
\end{aligned}
$$

The result matrix (the logically expanded relation S) now encodes two newly inferred data triples: `(car101 below sky101)` and `(shoes100 below hat100)`. Once again, this is precisely what conventional OWL reasoning would infer in the same situation.

Note that when the ordered pair for the counterpart axiom, `(below owl:inverseOf above)`, is encountered and processed as part of data inference, the same things happen, but since the order of the relations is now reversed, it is the matrix for relation `above` that acquires extra, inferred triples. Since OWL construct `owl:inverseOf` is symmetric, inference must be performed in both directions, *e.g.*, (above, below) and (below, above), before it can be considered to be fully completed.

Finally, recall that, with the axiom inference for `owl:inverseOf`, to perform inference for *symmetry* we used only the relational *converse* operation followed by the relational *union* operation. Observe that here, with data inference for `owl:inverseOf`, where we want to infer actual *inverses*, we use the same two relational operations, in the same order. In fact, we can see that the mechanics for inferring *symmetry* and *inverses* are remarkably similar. In both cases, we start by finding the converse of some relation (*i.e.*, we flip the coordinates of the ordered pairs of that relation). The only question is where we *union* that converse relation. For inverses, we *union* the converse relation with the inverse relation. For symmetry, we *union* the converse relation with the relation itself. Thus, a useful way of understanding symmetry is to see it as being a special case

of inverses. This is not a new insight; it can be found in Allemang et al. (2020), for example. Our tensor knowledge graph reasoning for OWL property `owl:inverseOf` showcases the usefulness of this perspective, and makes it concrete.

5.3.3 Emulating rdfs:domain inference semantics

The inference semantics associated with RDFS property `rdfs:domain` are captured in the following rule (whose expression has again been tailored for our forthcoming example):

```
if (eat rdfs:domain Mammal) and (person eat pizza),
then (person rdf:type Mammal)
```

That is, if we declare a domain for user-defined property, then, whenever an OWL reasoner sees an individual playing the 'subject' role in a triple with property `eat`, it will infer that individual is of type `Mammal`, *i.e.*, is a member of the class `Mammal`. We step through a concrete example of how our tensor knowledge graph reasoning techniques allow us to emulate these inference semantics.

Ontology and data The axiom `(eat rdfs:domain Mammal)` of mini VRD-World declares that user-defined object property (*i.e.*, binary relation) `eat` has its domain restricted to the class `Mammal`. For use in forthcoming examples, let the following set of axioms declaring domain restrictions constitute the `rdfs:domain` binary relation:

```
 eat rdfs:domain Mammal
ride rdfs:domain Person
wear rdfs:domain WearCapableThing .
```

Similarly, let the following two data triples constitute the binary relation `eat`:

```
person100 eat pizza100
   cat101 eat pizza101 .
```

Tensor knowledge graph instantiation During instantiation of the tensor knowledge graph for our mini VRD-World ontology, the axiom `(eat rdfs:domain Mammal)` is encoded in the $P \times C$ binary matrix representing the binary relation `rdfs:domain`. Recall that P is the number of user-defined object properties, and C the number of classes. This matrix occupies a particular channel of the $2 \times P \times C$ Tensor E of our tensor knowledge graph.

For example, let the ordered list of property names for mini VRD-World have size $P = 4$ and look like this: [`eat`, `has`, `ride`, `wear`]. Similarly, let the ordered list of class names have size $C = 6$ and look like this: [`Cat`, `Food`, `Mammal`, `Person`, `Sky`,

WearCapableThing]. Given these conditions, and given the rdfs:domain binary relation specified above, the $P \times C$ matrix encoding binary relation rdfs:domain, denoted D, is 4×6 and given by

$$D = \begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{pmatrix}.$$

During tensor knowledge graph instantiation, the data triples using property eat are encoded in an $N \times N$ binary matrix occupying a particular channel of the $P \times N \times N$ Tensor C of our tensor knowledge graph (where N is the number of individuals). For a concrete example, let the ordered list of individual names for mini VRD-World have size $N = 6$ and look like this: [cat101, dog100, person100, pizza100, pizza101, sky100]. Given this condition, and given the binary relation for eat specified above, the $N \times N$ matrix encoding binary relation eat, denoted E, is 6×6 and given by

$$E = \begin{pmatrix} 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Data inference In the case of property rdfs:domain, no *axiom inference* is required, so we jump straight to performing data inference. For emulating rdfs:domain inference, we use a relational *composition* operation followed by a relational *union* operation. Recall that we implement relational composition with matrix multiplication (followed by clamping at 1).

Our objective is to infer triples like (x rdf:type C). Triples for property rdf:type are encoded in the single binary matrix of Tensor D, which is $1 \times N \times C$. Thus, if we are to use relational composition, we know that our matrix multiplication needs to produce a matrix of size $N \times C$ (with the right contents) that we can then *union* with the existing $N \times C$ matrix for property rdf:type.

We also know that the binary matrix encoding relation rdfs:domain has shape $P \times C$. It clearly must feature as one of the two relations involved in our relational composition operation. Thus, if we are to leverage relational composition for rdfs:domain inference, we know that we need relations whose binary matrix representations have the following sizes:

$$(N \times P) \times (P \times C) = (N \times C). \tag{5.1}$$

To make our procedure for emulating `rdfs:domain` inference clearer, let us give names to the relevant matrices so that we can refer to them in relational operation equations. First, let T (for type) denote the $N \times C$ matrix from Tensor D, described above, that encodes the `rdf:type` relation. Let S (encoding a subjectOf relation) denote the $N \times P$ matrix in the above equation. We explain where S comes from shortly. Recall that we use D (for domain) to denote the $P \times C$ matrix encoding the `rdfs:domain` relation. And let T_D denote the $N \times C$ matrix that results from the relational composition operation expressed above. Given these matrix names, we can express how we emulate the inference semantics of OWL property `rdfs:domain` with the following procedure, consisting of the relational composition operation

$$S \odot D = T_D,$$

followed by the relational union operation

$$T \leftarrow T_D \cup T.$$

Recall that matrix T encodes the `rdf:type` binary relation that maps individuals to their classes. The inference semantics associated with OWL construct `rdfs:domain` can infer additional mappings of individuals to classes, captured in matrix T_D. If T_D contains new mappings, these represent a logical expansion of the `rdf:type` relation, and we integrate this expansion via the relational union operation.

Notice that all three of the matrices (relations) in Equation 5.1 are *heterogenous* relations, and that their shapes will generally be rectangular, but can be square. We have our $P \times C$ matrix already: matrix D encoding relation `rdfs:domain`. But the $N \times P$ matrix we denoted by S has no obvious source. There are no relations whose matrices have shape $N \times P$ anywhere within our tensor knowledge graph specification. Must we abandon our ambition of using relational composition here? No.

We can *derive* the precise relation we need by manipulating the whole of Tensor C of our tensor knowledge graph, which has shape $P \times N \times N$. The information we need is: which individuals appear in the 'subject' role of the triples for each user-defined property. That is, we want the *domain* of each user-defined property, *i.e.*, the *domain* of each of the P user-defined relations. We call the $N \times P$ relation we need `subjectOf`, and the triples it encodes will share the pattern `(individualX subjectOf propertyY)`. The information we need is in Tensor C, we just need to derive it.

The middle dimension of $P \times N \times N$ Tensor C corresponds to individuals appearing in 'subject' roles, and the last dimension to individuals appearing in 'object' roles. To make this clear, we add subscripts and refer to Tensor C as having shape $P \times N_s \times N_o$, and remember that $N_s = N_o$. To derive the $N \times P$ `subjectOf` relation we need from Tensor C, we do the following:

1. we permute the shape of Tensor C such that $P \times N_s \times N_o$ becomes $N_o \times P \times N_s$, which puts the individuals playing an 'object' role in triples up front in the first dimension
2. we then abstract-away the first dimension by perform a large relational *union* operation across the N_o channels of this 3D volume; thus, the $N_o \times P \times N_s$ tensor becomes a $P \times N_s$ matrix
3. finally, we perform a relational *converse* operation which transforms the $P \times N_s$ matrix into an $N_s \times P$ matrix (*i.e.*, we take the transpose).

Now that we have our $N \times P$ `subjectOf` relation, we have everything we need to perform our relational *composition* operation. The $N \times C$ matrix that results from the relational composition operation contains our inferred triples, like `(person100 rdf:type Mammal)` and `(cat101 rdf:type Mammal)`. Lastly, we perform a relational *union* operation on this $N \times C$ matrix of inferred triples in order to union it with the existing $N \times C$ relation stored in Tensor D of our tensor knowledge graph.

Data inference example Here we walk through a concrete example illustrating the inference procedure described above. First, let the `rdf:type` relation be given by the following set of data triples:

```
   cat101 rdf:type Cat
   dog100 rdf:type Mammal
person100 rdf:type Person
 pizza100 rdf:type Food
 pizza101 rdf:type Food
   sky100 rdf:type Sky
```

Given the ordered lists of individual names and class names specified earlier, and given the `rdf:type` relation just specified, $N \times C$ matrix T is 6×6 and given by

$$T = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \end{pmatrix}.$$

Next, consider S, the $N \times P$ matrix derived from the 3D volume of Tensor C of our tensor knowledge graph. If we assume the ordered lists of individual names and property names specified earlier, and we focus on encoding only the subjects of the binary

relation `eat` (as given, above, by the two data triples having predicate `eat`), then S is 6×4 and given by

$$S = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

To interpret matrix S, note that the columns correspond to the ordered list of property names: [`eat`, `has`, `ride`, `wear`]. The rows correspond to the ordered list of individual names: [`cat101`, `dog100`, `person100`, `pizza100`, `pizza101`, `sky100`]. Column 1 of the matrix encodes the *domain* of relation `eat`, *i.e.*, the individuals that appear as subjects in triples where `eat` is the predicate. Per the two data triples that constitute the `eat` relation, defined above, there are two such subject individuals: `cat101` and `person100`. Column 2 encodes the *domain* of relation (predicate) `has`, and indicates that `dog100` and `person100` appear as subjects, *e.g.*, in hypothetical data triples such as `(dog100 has pizza100)` and `(person100 has pizza101)`. Columns 3 and 4, for relations (predicates) `ride` and `wear`, respectively, are empty because no reasonable triples involving these predicates can be hypothesised given the particular individuals in our restricted set of 6. This also helps to declutter the matrix, which makes the relational composition step easier to follow.

Finally, for matrix D (encoding binary relation `rdfs:domain`), we can reuse the example 4×6 matrix defined earlier. We can now work through our inference procedure using the matrices we have constructed for our example scenario. First, we perform the relational composition operation to infer mappings between individuals and classes that are implied by `rdfs:domain` axioms, giving the matrix we named T_D:

$$S \odot D = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = T_D.$$

Next we perform the relational union operation between matrices T_D and T, so that any newly inferred mappings between individuals and classes are captured and duly encoded

in a (logically expanded) `rdf:type` relation.

$$T_D \cup T = \begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \cup \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \end{pmatrix}.$$

We can see that two new mappings have been inferred and added to the matrix that encodes the binary relation `rdf:type`: one encodes `(cat101 rdf:type Mammal)`, and the other encodes `(person100 rdf:type Mammal)`. These are precisely the inferences that conventional OWL reasoning would make given the same scenario.

A comment on efficiency and scalability of reasoning We have just illustrated our tensor knowledge graph reasoning approach to emulating the inference semantics of OWL property `rdfs:domain`. This illustration provides a good opportunity to highlight an important observation with respect to our tensor knowledge graph reasoning engine in relation to reasoning efficiency and scalability. One important strength, or advantage, of our tensor knowledge graph approach to symbolic OWL reasoning is that much (but not all) of the computational burden associated with our reasoning techniques is *not* effected by the size of the graph. For example, for binary matrices of a given size, the computational burden of operations like matrix addition and matrix multiplication stays constant, regardless of how densely those matrices may be populated with 1s rather than 0s. Thus, for instance, the derivation of the `subjectOf` relation described above, where an $N \times P$ binary relation matrix is derived from the $P \times N \times N$ binary Tensor C, will take the same amount of computation now matter how densely populated Tensor C is with 1s (which is a direct reflection of the volume of data triples in the graph that connect the N individuals). The derivation of relation `subjectOf` is already efficient; but it also scales beautifully.

5.3.4 RDFS and OWL inference rule coverage

Here we state the aspects of RDFS and OWL inference semantics currently supported by our tensor knowledge graph reasoning techniques. We continue to use our distinction between *axiom* inference and *data* inference because we find it helpful. But this informal distinction of ours is foreign to the RDFS and OWL specifications.

Coverage of RDFS inference semantics With respect to the RDFS entailment rules (see Section 9.2 of Hays and Patel-Schneider, 2014), here we confirm which

RDFS rules are currently supported by our tensor knowledge graph reasoning techniques, and which not. The RDFS entailment rules are numbered, following the pattern `rdfsX`.

RDFS axiom inference covered:

- `rdfs:subClassOf` transitivity (per rdfs11)
- `rdfs:subPropertyOf` transitivity (per rdfs5)

RDFS data inference covered:

- `rdfs:subClassOf` proper (per rdfs9)
- `rdfs:subPropertyOf` proper (per rdfs7)
- `rdfs:domain` (per rdfs2)
- `rdfs:range` (per rdfs3)

RDFS entailment rules not currently covered:

- `rdfs:subClassOf` reflexivity (per rdfs10)
- `rdfs:subPropertyOf` reflexivity (per rdfs6)
- rdfs1 - not consequential enough (yet)
- rdfs4a - not consequential enough (yet)
- rdfs4b - not consequential enough (yet)
- rdfs8 - not consequential enough (yet)
- rdfs12 - not consequential enough (yet)
- rdfs13 - not consequential enough (yet)

With respect to reflexivity, our techniques can quite easily support RDFS entailment rules rdfs6 and rdfs10 because inferring reflexivity when a binary relation is represented as a binary matrix is as simple as putting 1s down the diagonal. But we do not yet do this because it creates a lot of clutter without adding much value.

Coverage of OWL inference semantics Here we state the aspects of OWL inference currently supported by our tensor knowledge graph reasoning engine. For the formal specification of the inference semantics of OWL properties, see Motik, Patel-Schneider, et al. (2012). The aspects of OWL not yet covered are too many to be usefully listed.

OWL axiom inference covered:

- `owl:equivalentClass` symmetry (but not reflexivity, yet)
- `owl:equivalentProperty` symmetry (but not reflexivity, yet)
- `owl:inverseOf` symmetry

OWL data inference covered:

- `owl:inverseOf`
- user-defined properties declared as symmetric
- user-defined properties declared as asymmetric
- user-defined properties declared as transitive
- user-defined properties declared as reflexive
- user-defined properties declared as irreflexive

Note that the inference semantics associated with user-defined properties that are declared to be *asymmetric* is different in nature to most of the inference semantics we have discussed so far. If a property is declared to be *asymmetric*, this means it is unidirectional. That is, if property `p` is asymmetric, then if two triples such as `(s, p, o)` and `(o, p, s)` ever arise (either through assertion or inference), this represents a *violation* of asymmetry, a logical inconsistency. No new triples are ever inferred in relation to asymmetry itself. Declaring an OWL user-defined property to be asymmetric simply enables 'consistency checking' for that property, in order to catch cases where the condition of asymmetry is violated. This is the case in conventional OWL-based knowledge graphs and in our tensor knowledge graph reasoning engine.

Declaring a user-defined property to be *irreflexive* also leads to a consistency check rather than to the inference of new triples. In our case, with our binary matrix representations of OWL binary relations, checking for violations of the *irreflexive* characteristic is as simple as checking for 1s on the diagonal of the matrix.

The user-defined property characteristics of 'functional' and 'inverseFunctional' have not yet been fully implemented because their inference semantics are entwined with those of the `owl:sameAs` property, which declares that two individuals (with different names) are in fact the same. The inference semantics of `owl:sameAs` are surprisingly intricate and have been implemented only partially.

Equivalence to RDFS-Plus In their book on OWL for the working ontologist, Allemang et al. (2020) refer to a subset of OWL they call *RDFS-Plus*. Within RDFS-Plus they include all of RDFS, plus a handful of (what they regard to be) the most

commonly used and generally useful aspects of OWL. The aspects of OWL that they include in their definition of RDFS-Plus are these:

- `owl:inverseOf`
- symmetric user-defined properties
- transitive user-defined properties
- `owl:equivalentClass`
- `owl:equivalentProperty`
- `owl:sameAs`, together with functional user-defined properties and inverseFunctional user-defined properties
- and some other bits of less interest.

If we compare this list with the inference semantics coverage reported above, we can see that our tensor knowledge graph reasoning already matches the Allemang et al. (2020) definition of RDFS-Plus, except for support for `owl:sameAs` which is only partially implemented at this stage.

GraphDB ("Ontotext GraphDB", 2023), the OWL-based knowledge graph triple store product that we used in thread two of our research, as a symbolic reasoning engine, has a similar concept of *RDFS-Plus*. GraphDB organises its inference semantics capabilities into what it calls *rule sets*. One rule set is called *RDFS-Plus*. Their documentation states the the RDFS-Plus rule set supports RDFS, plus `owl:inverseOf`, symmetric user-defined properties and transitive user-defined properties. Their RDFS-Plus rule set is, in fact, the GraphDB default rule set. If we compare the GraphDB definition of RDFS-Plus with the inference semantics coverage reported above, we can see that our tensor knowledge graph reasoning already *exceeds* the GraphDB concept of RDFS-Plus.

In other words, our tensor knowledge graph reasoning capability, even at this early stage, is already non-trivial. We can say that the reasoning capability of our tensor knowledge graph reasoning engine is equivalent to that of *RDFS-Plus* (within the current limitations of our tensor knowledge graph representation).

5.3.5 The Roy-Warshall transitive closure algorithm

Here we discuss the first of two matrix-based transitive closure inference algorithms that we employ within our tensor knowledge graph reasoning engine to do the heavy-lifting associated with transitivity inference. The biggest computation burden in our tensor knowledge graph reasoning approach, by a long margin, is that associated with

handling transitivity. An OWL ontology may be rife with transitive relations, each of which requires their transitive closure to be inferred. The built-in OWL property `rdfs:subClassOf`, which is used to declare a class hierarchy in virtually every ontology, is transitive. The built-in OWL property `rdfs:subPropertyOf`, which is used to declare hierarchical relationships amongst user-defined properties, is transitive. And any or every user-defined property in an OWL ontology may be declared to be transitive. For our tensor knowledge graph reasoning engine to be viable in practical settings, therefore, it must have robust and efficient algorithms for inferring transitive closures. Hence, we have devoted special attention to this topic of symbolic inference. In this section we introduce and discuss the Roy-Warshall transitive closure algorithm. The Union-of-Powers transitive closure algorithm is discussed in the following section.

Regarding notation: throughout our discussions of transitive closure algorithms we use R to denote a (homogeneous) binary relation that is transitive. We follow Gross et al. (2019) by using a $*$ superscript to denote a transitive closure matrix. Hence, we say R^* is the transitive closure of transitive binary relation R. If a relation R is transitive, then to infer everything that is entailed by that transitive relation, we must find its transitive closure.

Algorithm 1 shows the Roy-Warshall transitive closure algorithm (Schmidt & Ströhlein, 1993; Warshall, 1962). This algorithm emerged within the field of graph theory, not relation theory. The algorithm is expressed with respect to an $n \times n$ Boolean matrix R. It turns R into R^*, the transitive closure of R. The algorithm, as expressed, performs $O(n^3)$ Boolean $\vee$ operations.

Algorithm 1 The Roy-Warshall transitive closure algorithm

```
1: for i = 1 to n do
2:    for j = 1 to n do
3:       if R(j, i) then
4:          for k = 1 to n do
5:             R(j, k) ← R(j, k) ∨ R(i, k)
6:          end for
7:       end if
8:    end for
9: end for
```

What it does is this:

- the outer two `for` loops, in lines 1 and 2, walk the grid
- in line 3, if the current cell $R(j, i)$ is `True`, then the inner `for` loop, in lines 4 and 5, does a *logical union* of row i into row j.

We adapted this algorithm only marginally for our tensor knowledge graph reasoning engine. In our version, line 3 checks if $R(j, i) > 0$, and line 5 uses scalar addition. We explored three variations. In variation *(i)*, we clamp each scalar addition result at 1, as part of line 5. In variation *(ii)*, we clamp the whole matrix at 1, once, at the end of the algorithm. Obviously, variation *(ii)* allows the values in the matrix to grow larger than 1, and in an uncontrolled way. Such a matrix is no longer a binary matrix. But, interestingly, this did *not* corrupt the algorithm in any way: variations *(i)* and *(ii)* deliver identical results, albeit with different speeds. Clamping once at the end is much faster, as one might expect. In variation *(iii)*, in line 5 we use logical disjunction ($\vee$) on the numeric values of the matrix. Again, variation *(iii)* delivered identical results.

5.3.6 The Union-of-Powers transitive closure algorithm

The second transitive closure algorithm we use in our tensor knowledge graph reasoning is one for which we have not encountered an official name or moniker. So that we can talk about easily, we call it the *Union-of-Powers* algorithm. We express the algorithm with respect to a Boolean matrix R that encodes some transitive binary relation, where R^* denotes the transitive closure. Equation 5.2 shows the algorithm.

$$R^* = \bigcup_{i=1}^{n-1} R^i \tag{5.2}$$

As the algorithm suggests, one takes successive powers of the Boolean matrix, using Boolean matrix multiplication, and then a *logical union* is performed over this series of 'power' matrices. The algorithm works because, as explained in Bang-Jensen and Gutin (2009), the matrix for the ith power of R, R^i, gives all of the paths (transitive chains) of length i.

The origins of the *union-of-powers* algorithm appear mixed. Schmidt and Ströhlein (1993) define the algorithm in the context of binary relations. Bang-Jensen and Gutin (2009) discuss the algorithm in the context of graph theory and adjacency matrices. For tensor knowledge graph reasoning, we adapt this algorithm only to the extent that we use binary matrices and standard matrix multiplication and clamping at 1, rather than Boolean matrices.

We can also understand this algorithm from the perspective of relational composition and transitivity, as mentioned earlier, in Section 5.3.1, when we introduced the relational composition operation. The matrix multiplication represents relational composition, but since there is only one relation, R, and it is being repeatedly composed with itself, the algorithm corresponds to the special case of relational composition, which is transitivity.

The *Union-of-Powers* algorithm is elegant, especially when compared with the Roy-Warshall algorithm. But its complexity is $O(n^4)$, which is concerning. A naive implementation, that computes all $n-1$ power matrices, may not scale well for large n.

5.3.7 Union-of-Powers stopping conditions

We studied the behaviour of the *Union-of-Powers* algorithm closely because we were certain that it had real potential to help us with our tensor knowledge graph reasoning. We were motivated by the observation that, to find a full transitive closure R^* of an $n \times n$ relation R, the worst-case, upper-bound number of iterations of the algorithm (the worst-case number of matrix multiplications), $n-1$, would only ever be necessary in very rare special cases that would (virtually) never arise in practice. These special cases are ones where the $n \times n$ relation R (or its counterpart directed graph with n nodes) contains paths (transitive chains) of length $n-1$. This equates to a directed graph of n nodes, where every node is connected in one long sequential chain. Graphs (relations) like this will rarely arise in practice. So a naive implementation of Union-of-Powers would be wasteful almost always.

In our implementation, we union the power matrices R^i as we go. So with each iteration i, we compute a power matrix R^i, and then we compute what we call the *union* matrix, $U^i = \bigcup_{k=1}^{i}(R^k)$, the union of everything so far, which is the emerging transitive closure. The union matrix U^i gradually becomes more dense with 1s as the power matrix R^i infers new transitive paths in the relation R. By examining the power matrix R^i, and the union matrix U^i, in each iteration, we discovered that the algorithm behaves like a dynamical system which, depending on the initial conditions (in our case, matrix R), either converges, diverges, or finds a stable equilibrium. More specifically, we discovered that, for every binary matrix R (simulating a random transitive binary relation) in our experiments, one of three conditions *always* arose:

(i) the *power* matrix R^i became all 0s (the empty relation), and the *union* matrix U^i stabilised (froze)

(ii) the *union* matrix U^i saturated and became all 1s (the universal relation)

(iii) sets of successive *power* matrices $R^j, R^{j+1}, \ldots, R^{j+k}$ evidenced a repeating pattern, and the corresponding *union* matrices $U^j, U^{j+1}, \ldots, U^{j+k}$ stabilised (froze), indicating that nothing *new* was being inferred, even though successive power matrices were still inferring new transitive paths (*i.e.*, were not all 0s).

To complete our analogy with dynamical systems: we regard behaviour *(i)* as akin to convergence, behaviour *(ii)* as akin to divergence, and behaviour *(iii)* as akin to a stable equilibrium. We adapted the *Union-of-Powers* transitive closure algorithm to create what we call our 'early-stopping Union-of-Powers' algorithm. In this implementation,

Table 5.1: Union-of-Powers Early-Stopping Analysis - For Binary Matrices of Size 500

Matrix Size	Number of Trials	Mean Matrix Density of 1s (%)	Union Matrix is all 1s	Power Matrices Repeating a Pattern	Power Matrix is all 0s	Mean Stopping Power	Mean Stopping Power as % of Size
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
500	1000	50.00226	1000	0	0	3.0	0.60
500	1000	40.12545	1000	0	0	3.0	0.60
500	1000	30.85704	1000	0	0	3.0	0.60
500	1000	22.66090	1000	0	0	3.0	0.60
500	1000	15.86309	1000	0	0	3.5	0.70
500	1000	10.56502	1000	0	0	4.0	0.80
500	1000	6.68275	1000	0	0	4.0	0.80
500	1000	4.00555	1000	0	0	5.0	1.00
500	1000	2.27603	985	15	0	6.0	1.20
500	1000	1.22281	107	893	0	8.2	1.63
500	1000	0.62116	0	1000	0	13.9	2.78
500	1000	0.29726	0	1000	0	32.3	6.47
500	1000	0.13508	0	665	335	11.4	2.29
500	1000	0.05796	0	275	725	5.4	1.08
500	1000	0.02278	0	107	893	3.5	0.71
500	1000	0.00989	0	41	959	3.0	0.61
500	1000	0.00056	0	17	983	3.0	0.60
500	1000	0.00000	0	3	997	2.9	0.59
500	1000	0.00000	0	3	997	2.6	0.52
500	1000	0.00000	0	1	999	2.2	0.45
500	1000	0.00000	0	0	1000	2.1	0.42
500	1000	0.00000	0	0	1000	2.0	0.40
500	1000	0.00000	0	0	1000	2.0	0.40

we check for these three conditions and use them as early-stopping conditions, to deliver (what we believe to be) a full transitive closure R^* without any wasteful (redundant) matrix multiplication computation.

We conducted a series of trials with our early-stopping Union-of-Powers algorithm whereby we would randomly instantiate a binary matrix R of a certain size, while arranging for that matrix to have a certain density of 1s. That is, the starting conditions are randomly generated, but controlled. We would then run matrix R through our adapted early-stopping Union-of-Powers algorithm and track which (if any) early-stopping condition gets triggered, along with the power number i, per R^i, at which early-stopping gets triggered.

Table 5.1 summarises our results for the trials we conducted with binary matrices R of size 500×500. It reveals an attractive symmetry in the behaviour of the Union-of-Powers algorithm, and it highlights the *value* of our three early-stopping conditions. Binary matrices R that are relatively dense with 1s (from 50% density down to 4% density, per column 3) *all* result in the *union* matrix U^i (column 4) saturating and becoming

all 1s. This triggers our early-stopping condition (ii). And we can see in column (7) how quickly this early-stopping condition arises for these matrices: after just 3, 4 or 5 iterations of the algorithm, not the worst-case number of iterations, $n - 1$, *i.e.*, $500 - 1 = 499$. Binary matrices R with densities of 2% down to about 0.02% tend to trigger our early stopping condition (iii), where the successive power matrices R^i fall into a repeating pattern (per column 5). As can see from column (7) for these matrices, stopping condition (iii) takes longer to arise. The number of iterations of the algorithm before early-stopping kicks-in rises steeply for these matrices R, reaching a peak mean of 32.3 iterations, before falling back again. But note that, per column (8), this peak mean of 32.3 iterations represents only 6.47% of the worst-case number of iterations. Finally, binary matrices R with densities of 1s below 0.05% increasingly result in the power matrix R^i becoming all 0s, per column (6). This triggers our early stopping condition (i). We see from column (7) that stopping condition (i) arises quickly, after only 2 or 3 iterations of the algorithm, not 499. Thus, presuming (for now) that our 'early-stopping Union-of-Powers' algorithm always delivers a full transitive closure, R^*, we can say that our three early-stopping conditions deliver important reductions in computation burden; reductions that make the Union-of-Powers algorithm appear viable, even attractive, despite its complexity of $O(n^4)$.

We ran analysis exercises identical to the one just described for binary matrices of size 250, 1000, and 5000. For each exercise, the results resemble those in Table 5.1, with the same attractive symmetry and all the same insights already discussed. We analysed the computational savings of our early-stopping conditions as a function of the size of matrix R. Table 5.2 reveals the counter-intuitive result that these savings actually *increase* (in relative terms) as the size of matrix R increases. Column (3) shows the peak mean stopping power number i, *i.e.*, the peak mean number of matrix multiplication operations that were needed for the matrices R of different size (*e.g.*, per column 7 of Table 5.1). We can see that these peak numbers of matrix multiplications increase with matrix size. This is not surprising; it feels intuitive. But column (4) shows that, in relative terms, these rising numbers of matrix multiplication operations represent ever *smaller* proportions of the worst-case number of matrix multiplication operations. This is evidence that our 'early-stopping Union-of-Powers' algorithm scales well.

The analysis just described shows that our three early-stopping conditions are effective at helping to make the Union-of-Powers transitive closure algorithm much more efficient than it would otherwise be. But all this time we have been presuming that our 'early-stopping Union-of-Powers' algorithm always delivers a full transitive closure R^* for any relation R. That is, we have presumed that our early-stopping conditions do not inadvertently stop too soon, thereby preventing the algorithm from inferring a full transitive closure R^*. We now examine this presumption for robustness.

We argue that stopping condition *(i)* is mathematically sound. Once power matrix R^i

Table 5.2: Union-of-Powers Early-Stopping Analysis - Computational Savings per Matrix Size

Matrix Size	Number of Trials	Peak Mean Stopping Power	Peak Mean Stopping Power as % of Size
(1)	(2)	(3)	(4)
250	1000	25.5	10.3
500	1000	32.3	6.47
1000	1000	43.6	4.36
5000	1000	84.4	1.69

becomes all 0s (the empty relation), this means there are no paths (no transitive chains) of length i in the relation (or directed graph). If there are no paths of length i, there can be no paths of length $i + 1$, and if there are no paths of length $i + 1$, there can be no paths of length $i + 2$, etc.. So nothing new will every be inferred, and hence it is safe to stop. We can reason differently, by observing that if power matrix R^i is all 0s, so will all subsequent power matrices, R^{i+1}, etc., because multiplication by 0 yields 0. Either way, stopping condition *(i)* is sound.

We argue that stopping condition *(ii)* is mathematically sound. Once the union matrix U^i, the emerging transitive closure, saturates and becomes all 1s, nothing new can possibly be inferred. The transitive closure is the universal relation, where everything maps to everything. It is safe to stop.

We are confident that stopping condition *(iii)* is mathematically sound, but we are not yet in a position to make a defensible argument. We can argue that repetitions of a pattern of power matrices, $R^j, R^{j+1}, \ldots, R^{j+k}$, will never infer anything new, as long as the pattern never ceases. That appears self-evident. We are confident that the source of the repeating patterns in the power matrices R^i are a consequence of *cycles* (closed paths) in the directed graph of the relation R. For example, suppose a cycle of length q in some directed graph. Then, in every qth power matrix, $R^q, \ldots, R^{2q}, \ldots, R^{3q}, \ldots$, q 1s will appear on the diagonal, each one connecting one of the q nodes in the cycle to itself. We suspect that algorithm behaviour *(iii)* may be entirely an artefact of cycles. If so, this strengthens our confidence that stopping condition *(iii)* is mathematically sound. In fact, we suspect that if this is indeed so, then our three early-stopping conditions likely collapse into one: if the *union* matrix U^i stops changing, then we can stop the algorithm, and be confident we have a full transitive closure.

Table 5.3: Transitive closure algorithm runtime comparison

Algorithm	Complexity	Operation	Mean Runtime (sec)
UoP (early stopping)	$O(n^4)$	matrix multiplication	0.002
UoP (naive)	$O(n^4)$	matrix multiplication	0.030
Roy-Warshall	$O(n^3)$	Boolean OR two scalars	18.506
Roy-Warshall	$O(n^3)$	add scalars, clamp matrix at end	29.973
Roy-Warshall	$O(n^3)$	add scalars, clamp each sum	37.215

5.3.8 Comparing Roy-Warshall with Union-of-Powers

To see how the two transitive closure algorithms, Roy-Warshall and Union-of-Powers, compare speed-wise, we conducted a small benchmarking exercise on a Mac Studio M2 Ultra machine. We generated a single binary matrix R of size 250×250, randomly, while arranging for its density in 1s to be in the range 1%. We did this because we did not want the transitive closure to saturate and become all 1s, per column (4) of Table 5.1. We then ran five trials for each of five algorithm variations: two variations of Union-of-Powers algorithm, and three of Roy-Warshall. We ran the trials in pairs, one Union-of-Powers algorithm and one Roy-Warshall algorithm. We did this so we could compare the transitive closures inferred by the two algorithms with one another, as a quality check. No discrepancies were ever identified. Table 5.3 shows the mean runtimes, in seconds, for each of the algorithms. The 'Operation' column shows the salient computation operation being performed by each algorithm.

One observation to make per Table 5.3 is that, with respect to the complexity of the two algorithms, $O(n^4)$ and $O(n^3)$, respectively, the runtime results are highly counter-intuitive. One would expect an $O(n^4)$ algorithm to be slower than an $O(n^3)$ algorithm, not orders of magnitude faster. The message appears to be that, given modern computing chips that have been optimised for matrix operations, if an algorithm depends on matrix multiplication, then its complexity (measured by conventional means) may well be an unreliable guide as to how to regard the algorithm relative to competing algorithms. Based on these results, we prefer the $O(n^4)$ Union-of-Powers algorithm over the $O(n^3)$ Roy-Warshall algorithm for performing transitive closure inference in our tensor knowledge graph reasoning engine.

Another observation from Table 5.3 is that our early-stopping Union-of-Powers algorithm is far faster than the naive Union-of-Powers algorithm (the one that computes the worst-case number of power matrices, R^i, all the way up to $i = n - 1$). This was expected, and here we see it confirmed. In the context of these trials, the early-stopping Union-of-Powers algorithm is 15 times faster than the naive implementation. Hence, we will tend to prefer our early-stopping Union-of-Powers algorithm for our tensor knowl-

edge graph reasoning engine.

A final observation with respect to Table 5.3 is that it appears that doing a Boolean OR on two scalars is faster than adding two scalars and doing clamping, no matter how the clamping is handled. This points to a potential performance optimisation we might make in our tensor knowledge graph reasoning techniques at some point. And, finally, notice what is implied by the variation of Roy-Warshall where we add two scalars but clamp the matrix to 1 only at the end of the algorithm. This implies that the values in matrix R (the relation whose transitive closure is being computed) are allowed to grow larger than 1. That is, this algorithm variation allows matrix R to become non-binary. And yet the algorithm is not corrupted in any way, and generates a transitive closure matrix identical to all the other algorithms. This is because our implementation checks for nonzero values rather than values of 1.

5.4 Applications

Up to this point our discussions have focused on concepts: tensor knowledge graphs, and tensor knowledge graph reasoning, based on relational mathematics. In this section, our focus shifts to applications of these concepts. We explore a variety of these. First we introduce the concept of a subsymbolic-symbolic neural network architecture and describe an instance of it we call the Combiner. The Combiner wraps a subsymbolic classifier and a symbolic module that can be used in different ways. Then we explore injecting symbolic knowledge encoded in a tensor knowledge graph. Next we examine having a purely subsymbolic network attempt to learn symbolic (binary) encodings of knowledge rather than injecting the knowledge. Then we introduce the concept of 'neural symbolic inference', which involves an instance of a subsymbolic-symbolic network where the symbolic module is equipped to perform actual symbolic logical inference, incrementally, within the rhythm of standard neural network forward pass. Finally, we discuss applications of the tensor knowledge graph in matrix factorisation knowledge graph embedding models.

5.4.1 The Combiner: a subsymbolic-symbolic architecture

Figure 5.1 shows a proof-of-concept neural network architecture that combines a subsymbolic learning module with a symbolic module. We think of this as a subsymbolic-symbolic neural network architecture. For our investigations, we focused on a particular category of subsymbolic-symbolic architecture, where the subsymbolic learning module is a classifier that predicts classes, and the symbolic module generalises those class predictions to their parent classes. More specifically, in our investigations the subsymbolic classifier is a conventional feed-forward, multi-class, single-label classifier. The

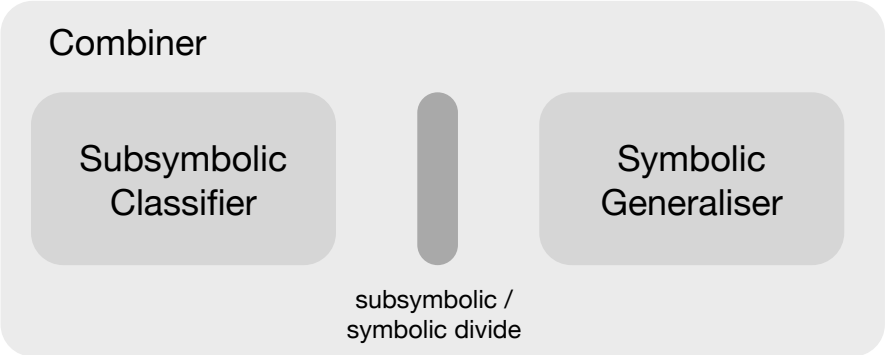


Figure 5.1: A conceptual instance of a generic subsymbolic-symbolic neural network architecture, in which a subsymbolic learning module is followed by a symbolic module. In this instance of the generic architecture, the role of the subsymbolic module is filled by a Classifier that predicts classes, and the role of the symbolic module is filled by a Generaliser whose job is to generalise the classes predicted by the Classifier to their correct parent classes (per an OWL class hierarchy), but by strictly symbolic means only. The subsymbolic / symbolic divide is the point of transition between the subsymbolic and symbolic worlds. These three conceptual components are wrapped within a larger neural network module referred to as a Combiner. The subsymbolic learning module and the symbolic module work in-concert with one another, within the rhythm of a conventional neural network forward-pass.

job of the symbolic module is to generalise the classes predicted by the classifier, in accordance with a class hierarchy defined in an OWL ontology, and by using strictly symbolic generalisation methods only. We refer to these two components as the *Classifier* and the *Generaliser*, respectively. We refer to a class predicted by a Classifier as a *base class*, and to generalised classes determined by a Generaliser as *parent classes*. The point of cross-over or hand-off between the subsymbolic learning module (in our case, the Classifier) and the symbolic module (in our case, the Generaliser) we refer to as the *subsymbolic / symbolic divide*. This is the point of transition between the subsymbolic and symbolic realms. All of these components are enclosed and sequenced within a larger, wrapping neural network module we refer to as the *Combiner*.

For our investigations, a forward-pass of the Combiner involves *(i)* executing the (conventional) forward-pass of the Classifier, *(ii)* crossing the subsymbolic / symbolic divide, by converting vectors of logits emitted by the classifier into corresponding binary, one-hot vectors—the symbolic representations of the base classes predicted by the Classifier, and *(iii)* executing the (conventional) forward-pass of the Generaliser, feeding it only binary one-hot vectors, the symbolic representations of base classes. For each forward-pass of the Combiner, the Combiner returns both the base classes predicted by the subsymbolic Classifier, and their corresponding parent classes, as determined by the symbolic Generaliser. The objective is for the Generaliser to, at all times, return the complete and correct set of parent classes for any given base class predicted by the

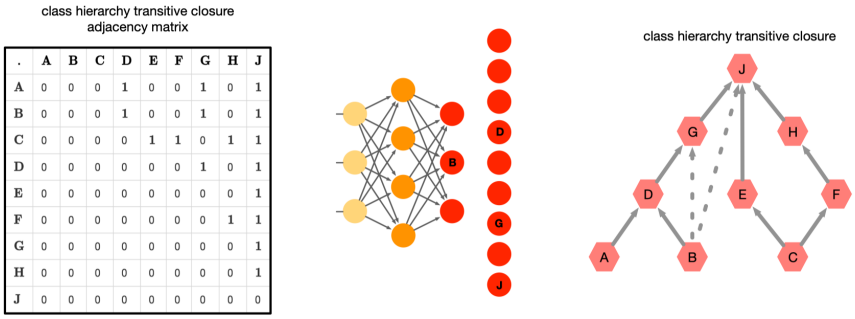
Classifier. The OWL ontology that defines the class hierarchy in question is the arbiter of truth.

The scenario just outlined is the backdrop for a series of investigations we conducted into ways in which tensor knowledge graphs and tensor knowledge graph reasoning capabilities can be leveraged by, and integrated in, subsymbolic-symbolic systems such as the one depicted in Figure 5.1. The three sections of this chapter that follow this one each describe one such investigation. In each investigation, the Classifier remains the same, but the Generaliser is adapted for the particular experiments at hand. Section 5.4.2 discusses an investigation of having the Generaliser do its job by looking-up injected symbolic knowledge. Section 5.4.3 discusses an extension of that investigation where the goal is to learn the encoded symbolic knowledge rather than injecting it. Section 5.4.4 describes a further extension where the Generaliser is adapted to infer the transitive closure of a class hierarchy whilst it infers parent classes.

5.4.2 Injecting symbolic class hierarchy knowledge

Here we describe an investigation that demonstrates that the symbolic knowledge of an OWL class hierarchy, encoded by a tensor knowledge graph, or by the tensor knowledge graph reasoning engine, can be *injected* into the Generaliser component of a subsymbolic-symbolic neural network (per Section 5.4.1). The investigation further demonstrates that the Generaliser can be equipped to efficiently retrieve the appropriate symbolic knowledge (*i.e.*, the correct parent classes for a given base class) from the knowledge base of injected knowledge. Figure 5.2 shows the key elements of this investigation. The three fully-connected neural network layers correspond to the Classifier component of our subsymbolic-symbolic model. The single, larger layer of neurons to its right corresponds to the Generaliser component of our subsymbolic-symbolic model. This is a linear layer with no biases and no activation function, and its size equals the number of classes in the OWL class hierarchy. For a complex OWL ontology class hierarchy, our tensor knowledge graph reasoning engine is an appropriate tool for consuming the OWL ontology (represented by the graph on the right of the figure) and for encoding the class hierarchy of the OWL ontology in a binary matrix (such as the one on the left of the figure). We assume, for now, that the tensor knowledge graph reasoning engine has been used to infer the transitive closure of an OWL class hierarchy, and that the binary matrix we obtain from it encodes this full transitive closure of the hierarchy.

Once obtained, the binary matrix that encodes the symbolic knowledge of the OWL class hierarchy transitive closure is injected into the weight matrix of the single linear layer of the Generaliser. At the same time, the weight matrix is configured to disable gradient computation, so that the injected symbolic knowledge is preserved. This 'symbolic weight matrix' now acts as a knowledge base. Because the Generaliser is fed only



.	A	B	C	D	E	F	G	H	J
A	0	0	0	1	0	0	1	0	1
B	0	0	0	1	0	0	1	0	1
C	0	0	0	0	1	1	0	1	1
D	0	0	0	0	0	0	1	0	1
E	0	0	0	0	0	0	0	0	1
F	0	0	0	0	0	0	0	1	1
G	0	0	0	0	0	0	0	0	1
H	0	0	0	0	0	0	0	0	1
J	0	0	0	0	0	0	0	0	0

Figure 5.2: Elements of an investigation into injecting symbolic knowledge encoding a class hierarchy into the Generaliser component of a subsymbolic-symbolic neural network architecture. The matrix on the left represents a binary matrix encoding of the class hierarchy on the right. The dotted links in the class hierarchy on the right are suggestive of the implied transitive closure of the hierarchy. The three fully-connected neural network layers in the middle represent the Classifier component of a subsymbolic-symbolic model. The single, larger layer of neurons to its right represents the Generaliser component of a subsymbolic-symbolic model. The binary matrix encoding the symbolic knowledge of the class hierarchy will ultimately be injected into the weight matrix of the single linear layer of the Generaliser, thereby enabling the Generaliser to find the parent classes of any base class predicted by the Classifier, using the normal matrix multiplication of a neural network forward-pass.

binary, one-hot vectors as input (*i.e.*, symbolic representations of base classes predicted by the Classifier), the normal matrix algebra of a conventional neural network forward pass allows the generaliser to retrieve all of the correct parent classes from its weight matrix knowledge base for any given base class predicted by the Classifier.

To see this, imagine a single binary one-hot vector as a row vector. If that binary one-hot row vector is multiplied against a matrix (any matrix), matrix algebra ensures that the result vector is equivalent to one of the rows of the matrix. In other words, if the row vector is binary one-hot, then the matrix multiplication effectively 'selects a row' from the matrix. In our case, because the matrix in question represents injected knowledge, the matrix multiplication can be viewed as akin to knowledge lookup or knowledge retrieval. A body of symbolic knowledge has been injected (a knowledge base), and the knowledge (of parent classes) is retrieved, selectively, repeatedly, as needed, driven by the base class predictions emerging from the Classifier. And, importantly, the Generaliser retrieves the correct parent classes from the injected knowledge base without resorting to any procedural code whatsoever. A symbolic class prediction (a binary one-hot vector) and forward-pass matrix algebra allow the knowledge retrieval to happen automatically.

The net result (or net benefit), is that the subsymbolic-symbolic model is able to perfectly emulate the symbolic reasoning (or class subsumption inference) that OWL rea-

soning would perform (using the same OWL ontology). As we demonstrated in thread two of our research, in Chapter 4, we could opt instead to call an OWL-based knowledge graph, as a symbolic reasoning engine, and have OWL reasoning perform the class generalisation for us, on-the-fly. Functionally, we know that would work; but it would be slower. By encoding the symbolic knowledge of the OWL class hierarchy (with the help of the tensor knowledge graph reasoning engine), and injecting that knowledge into a subsymbolic-symbolic network model, the model has the knowledge (of parent classes) it needs, at hand, and the generalisation occurs much faster.

An early form of the investigation just discussed is mentioned in Herron, Jiménez-Ruiz, and Weyde (2023), and again in Herron et al. (2025) at greater length. In the latter, we describe how our idea of encoding class hierarchy symbolic knowledge in a binary matrix can be leveraged in connection with Logic Tensor Networks (LTN) (Badreddine et al., 2022; Serafini & d'Avila Garcez, 2016). In the LTN setting that we imagine, the binary matrix that encodes a class hierarchy would participate in LTN *Real Logic* knowledge axioms (logical constraints), by being hosted within a *Real Logic* function. That function could provide fast class generalisation services, enabling the granular classes in the data to be quickly generalised into higher-level classes. The uplift in generality of the classes could then be leveraged to good advantage by helping to uplift the generality of the axioms (the logical constraints) themselves, thereby potentially reducing the number of axioms (constraints) that need be expressed. This idea generalises to other approaches to leveraging logical constraints in neurosymbolic systems, such as ROAD-R (Giunchiglia et al., 2023).

Variation: injection of partial knowledge Now we describe an important variation of the above investigation. As mentioned earlier, up to now we have assumed that the class hierarchy knowledge that is encoded in the binary matrix reflects the full transitive closure of the class hierarchy. This is what enables the Generaliser to retrieve *all* parent classes for a given base class in just one knowledge retrieval lookup, because each row of the weight matrix (the knowledge base) contains all of the parent classes for the corresponding base class. However, we can run the same experiments as above, and get the same results, if we opt, instead, to use a binary matrix that encodes only partial knowledge of the class hierarchy.

The partial knowledge is the class hierarchy as *asserted* in the OWL ontology, rather than its implied transitive closure. We can readily adapt the Generaliser for this setting. All we need do is arrange for the Generaliser to *loop* over its single layer until it has walked its way up the hierarchy, one parent class at a time, gathering the parent classes as it goes. This walk up the class hierarchy is accomplished without any intervention other than the looping. Once again, it is the combination of binary one-hot vectors and matrix algebra that ensures that the step-by-step walk up the class hierarchy, for any

given base class, visits all of the correct parents, and that this all happens automatically. We used a parameter to help the Generaliser know when it is safe to stop looping (*i.e.*, to know when it has reached the top of the class hierarchy). This parameter is the length of a *longest path* in the class hierarchy. The tensor knowledge graph reasoning engine can be equipped to provide the length of such a longest path.

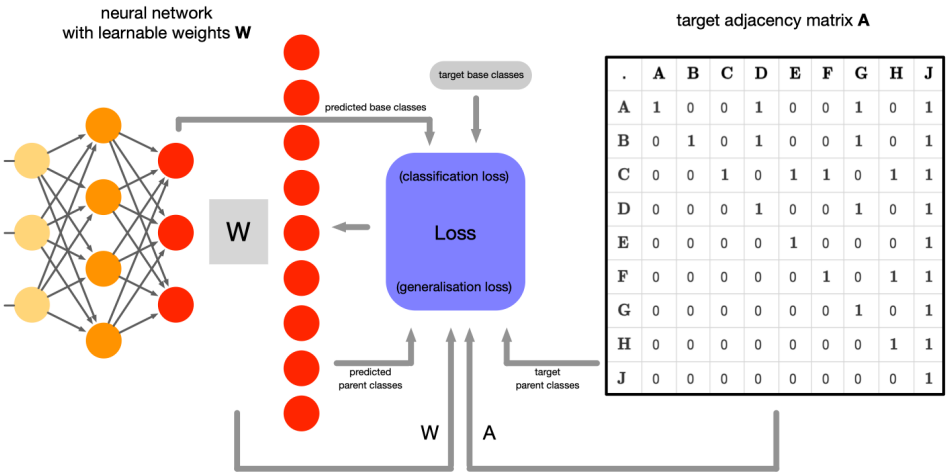
Observe that, in the original setting, the injected knowledge gave the Generaliser all of the parent classes for every base class, on a plate, ready to be looked-up in one go. Now, in this partial knowledge injection variation, the Generaliser has to find all the parents of a base class for itself, by traversing the hierarchy. Thus, in the setting of this variation, it is tempting to claim that the Generaliser is now performing symbolic *inference* rather than simply *knowledge retrieval*. But it is also reasonable to argue that, no, the Generaliser is not doing inference, it is simply performing repeated knowledge retrieval. We let readers decide for themselves how to interpret the Generaliser's role in this setting.

5.4.3 Learning symbolic class hierarchy knowledge

In this section, we discuss an investigation that builds upon the one described in the previous section. In this extension of the previous investigation, instead of *injecting* symbolic knowledge into the Generaliser component of the Combiner subsymbolic-symbolic model, the objective is to see if the Combiner is capable of *learning* the symbolic knowledge for itself. The idea here is to use the binary matrix that encodes the transitive closure of the class hierarchy as a target in conventional supervised learning, and then to check how well the network can predict the parent classes of base classes predicted by the Classifier component.

Observe, however, that, in this setting, the Combiner stops being a subsymbolic-symbolic neural network, per our definition in Section 5.4.1, and reverts to being a conventional subsymbolic neural network. The single linear layer of the Generaliser acquires biases, and an activation function; its weight matrix is configured to allow gradient computation so it becomes learnable, instead of being frozen; and the subsymbolic/symbolic divide disappears, because the logits emitted by the Classifier are no longer converted into binary one-hot vectors. But, despite this, the setting retains a neurosymbolic character in that one objective is to learn a binary matrix (which represents symbolic knowledge), and another is to learn a subsymbolic way of performing class generalisation, that can mimic symbolic (OWL) subsumption reasoning.

Figure 5.3 summarises the setting for this investigation. The Combiner, with its Classifier and Generaliser components, are on the left. The binary matrix on the right is the encoded class hierarchy that is used as a target during training. The objective is for the Combiner to learn that target binary matrix within the weight matrix of the single layer



.	A	B	C	D	E	F	G	H	J
A	1	0	0	1	0	0	1	0	1
B	0	1	0	1	0	0	1	0	1
C	0	0	1	0	1	1	0	1	1
D	0	0	0	1	0	0	1	0	1
E	0	0	0	0	1	0	0	0	1
F	0	0	0	0	0	1	0	1	1
G	0	0	0	0	0	0	1	0	1
H	0	0	0	0	0	0	0	1	1
J	0	0	0	0	0	0	0	0	1

Figure 5.3: A setting for investigating learning symbolic class hierarchy knowledge. The binary matrix on the right is a symbolic encoding of a class hierarchy. The network on the left is a classifier being trained to learn that binary matrix, within the weight matrix of last layer of the network. A two-part loss function is employed for this purpose. A classification loss is computed based on the predictions of base classes, and generalisation loss is computed as a function of the difference between the weight matrix under study and the target binary matrix used for supervision.

of the Generaliser. The computation of loss involves two components: a classification loss, with respect to the base classes predicted by the Classifier, and a generalisation loss, that is a function of the difference between the weight matrix of the Generaliser and the target binary matrix.

The same student at City St George's who we collaborated with on the investigation described in Section 5.4.2, drove the development and ran the experiments associated with the investigation described here. We do not have access to the results, so we cannot report how closely the network was able to learn the target binary matrix (the encoded symbolic knowledge), or how well the resulting model performed at generalising the base classes emitted by the Classifier component of the Combiner.

But our primary motivation here is to give examples of how the tensor knowledge graph reasoning engine can play useful roles in neurosymbolic research that involves the use of OWL-based knowledge graphs. The opportunity for the tensor knowledge graph reasoning engine to play a part in investigations such as the one described here remains the same as that for the knowledge injection investigation described in the preceding section. Something needs to create the binary matrix that encodes the transitive closure of the class hierarchy. The tensor knowledge graph reasoning engine does this. And the larger and more complex the class hierarchy of an OWL ontology, the more the value

of this particular aspect of the tensor knowledge graph reasoning engine's functionality will be appreciated.

5.4.4 Neural symbolic inference

In this section we discuss an investigation that builds upon the knowledge injection investigation described in Section 5.4.2. In this extension of that investigation, the goal is to demonstrate that a symbolic module in a neural network (like our Generaliser in our subsymbolic-symbolic Combiner) can perform proper symbolic logical inference within the rhythm of a neural network forward-pass. The symbolic logical inference problem we consider is that of inferring the transitive closure of a class hierarchy. We think of this investigation as demonstrating what we call *neural symbolic inference*. We avoid using the term neurosymbolic here, because the inference that our Generaliser performs is strictly symbolic.

We begin by introducing the idea of neural symbolic inference. Then we demonstrate that it works, by showing evidence from having had the Generaliser infer the transitive closure of our VRD-World OWL ontology. We close by describing an approach to crossing the subsymbolic/symbolic divide of our subsymbolic-symbolic neural network architecture differentiably.

The idea of neural symbolic inference

We introduce the idea of neural symbolic inference by describing the investigation we conducted in order to prove the idea. In this investigation, the Combiner retains its subsymbolic-symbolic character, as described in Section 5.4.1. We use *partial* knowledge injection, in the same way as described in Section 5.4.2. That is, we encode the *asserted* class hierarchy (not the transitive closure) in a binary matrix, and inject that partial knowledge of the class hierarchy into the weight matrix of the Generaliser (while disabling gradient computation). Unlike the investigation in Section 5.4.2, however, while the Generaliser's weight matrix is similarly immune to gradient descent, here it is not frozen and unchanging. In this setting, the Generaliser updates its own weight matrix (its knowledge base). It infers new knowledge and stores this new knowledge in its knowledge base, thus expanding it. In other words, its weight matrix (the emerging transitive closure of the class hierarchy) gradually becomes more and more dense with 1s.

With every forward-pass of the Combiner, the over-arching role of the Generaliser remains the same: to find and return the parent classes of each base class predicted by the Classifier. It always does accomplish this task, but how it does this, and the amount of work it takes, changes over time, as a consequence of the incremental logical inference it performs as part of each successive forward-pass. Recall that with partial knowledge

injection, as discussed in Section 5.4.2, the Generaliser must loop over its single layer in order to walk up the class hierarchy, hop-by-hop, to find parent classes. It must loop to the top of the class hierarchy to ensure it picks up every parent class of a base class. But once the transitive closure of the class hierarchy is fully inferred, there is no more need for looping, because all of the parent classes for every class are now explicitly identified in each row of the weight matrix (the knowledge base). So, in recognition of this, the Generaliser changes its behaviour: it stops doing inference and reverts to knowledge retrieval—a simple one-time look-up, just as we saw in Section 5.4.2.

The parallels with human behaviour here are worth highlighting. We usually start with partial knowledge of some topic. We reason and accumulate more knowledge on the topic, which we commit to memory. When we need that knowledge again (assuming we have not forgotten it), instead of re-reasoning we simply retrieve the knowledge from memory. In its own simplistic way, the Generaliser, in this 'neural symbolic inference' setting, mimics this behaviour.

To equip the Generaliser to infer the transitive closure of the class hierarchy encoded in its weight matrix, we adapted the Roy-Warshall transitive closure algorithm, described in Section 5.3.5, in Algorithm 1, that we had implemented within our tensor knowledge graph reasoning engine. The adaptations we applied were limited to fitting the algorithm to the neural network forward-pass setting of the Generaliser. Recall that the Roy-Warshall algorithm has two outer `for` loops that walk the grid of the binary matrix. And that within those two `for` loops, the algorithm does a logical union of one row into another. Within the context of inferring the transitive closure of a class hierarchy, we can describe this as doing a logical union of a parent row into a child row. We can see that, if this operation is performed enough, each row (each class) will eventually acquire all of its parents. This is the essence of the algorithm.

In the context of the Combiner, however, we do not walk a grid. Instead, we use the base classes predicted by the Classifier (rendered symbolically as binary one-hot vectors) in the forward-pass to drive the inference process. Thus, 'systematic walking of a grid' is replaced by 'seemingly random placement at a particular base class in the class hierarchy'. That is, we discarded the two outer `for` loops of Roy-Warshall and we let the incoming base class predictions determine the point in the class hierarchy at which the next bit of inference begins. The only other adaptation to Roy-Warshall we made was for efficiency. Remember that, as shown in Algorithm 1, Roy-Warshall handles the logical union of one row into another by walking the two rows cell-by-cell. We union at the level of rows, not cells.

As might be suspected, replacing 'systematic walking of a grid' with 'seemingly randomly occurring base class predictions that place us somewhere in the hierarchy' has implications with respect to being sure of inferring a full transitive closure. With our

refit of Roy-Warshall for neural networks, we cannot be sure. A full transitive closure of the class hierarchy may emerge within the course of a neural network training cycle, but it may not. It depends on how broadly and frequently the base classes predicted by the Classifier cover the space of the class hierarchy. The optimal conditions are for the set of base classes that the Classifier can possibly predict to map one-to-one with leaf classes in the class hierarchy. This way, the Roy-Warshall inference within the Generaliser at least is assured of potential access to every path in the class hierarchy. In such conditions, a full transitive closure will emerge if the Classifier predicts each of those base classes frequently enough.

The important thing is that the Generaliser is *capable* of inferring a full transitive closure; whether or not it does so depends on the classes in the training data, and how they map to the class hierarchy in question, and how frequently they are all predicted by the Classifier. And failing to infer a full transitive closure is a benign outcome. The Generaliser always returns the full set of parent classes, regardless of the state of its knowledge base (its weight matrix). The state of its knowledge base just determines how much work it has to do to get the parents. The absence of a full transitive closure is a condition invisible to the world outside the Generaliser.

The inference performed by the Roy-Warshall algorithm is equivalent to RDFS entailment rule `rdfs11` (Hays & Patel-Schneider, 2014), which declares that RDFS property `rdfs:subClassOf` is transitive. That rule can be expressed as follows:

```
if
    (A rdfs:subClassOf B) and (B rdfs:subClassOf C),
then
    (A rdfs:subClassOf C).
```

Variation: Union-of-Powers algorithm As a final word, we briefly point out that the Union-of-Powers transitive closure algorithm used in our tensor knowledge graph reasoning engine can also be adapted for use within our concept of neural symbolic inference. Recall that the algorithm relies on nothing but matrix multiplication (relational composition). One can readily imagine arranging for the Generaliser to perform, say, one matrix multiplication operation per forward-pass.

Demonstrating neural symbolic inference using VRD-World

To verify that our concept of neural symbolic inference works in more than toy settings, we tested it on our VRD-World OWL ontology, introduced in Chapter 3. The current limitations of our tensor representation of OWL-based knowledge graphs, described in Section 5.2.2, however, mean that our current tensor knowledge graph reasoning engine can not yet handle *all* of the class hierarchy declared in VRD-World. This prompted

us to create a special-purpose tool for encoding the class hierarchy of VRD-World. We called this tool the 'OWL class hierarchy adjacency matrix' tool, or the OCHAM tool, for short. The tool uses an OWL-based knowledge graph and OWL reasoning, so it is suitable for use with any OWL ontology, not just VRD-World. It generates a binary matrix that encodes the class hierarchy as asserted in an OWL ontology file, and a binary matrix that encodes the transitive closure of the class hierarchy.

Using OCHAM, we obtained the two binary matrices that we needed with respect to VRD-World. The binary matrix that encodes the asserted VRD-World class hierarchy is the partial knowledge of the class hierarchy that we injected into the Generaliser just after instantiation. The binary matrix that encodes the transitive closure of the VRD-World class hierarchy, as determined by OWL reasoning, was used as the arbiter of truth. Next we used OCHAM to give us the length of a longest path in our encoded class hierarchy, so that the Generaliser will know, whilst looping over its single layer, when it has reached the top of the class hierarchy and can stop looping. The length of a longest path in the class hierarchy of VRD-World is 9.

Since the Generaliser is only ever presented with binary one-hot vectors, we exploited this fact and *simulated* a Classifier, and thereby greatly simplified our testing effort. We instantiated a Generaliser, fitted with our adapted Roy-Warshall transitive closure inference algorithm, and fed it binary one-hot vectors corresponding to all of the object classes defined in the NeSy4VRD dataset. Each one of these classes corresponds to a particular leaf class in the class hierarchy of VRD-World. And there are no other leaf classes in the hierarchy. To simulate an epoch's worth of training, we arranged to do one iteration over the class names of NeSy4VRD and converted each one into a binary one-hot vector representation which was then passed to the Generaliser. At the start of each (simulated) epoch, we recorded the before-epoch state of the Generaliser's weight matrix (the encoded class hierarchy, and the emerging transitive closure of that class hierarch), and just after each epoch we compared the before-epoch state with the updated state and computed and reported the number of new inferences made by the Generaliser during that epoch.

At the end of each (simulated) epoch, we also compared the updated state of the Generaliser's weight matrix with the arbiter of truth: the binary matrix encoding the transitive closure, as inferred by OWL reasoning. We computed and reported the progress made by the Generaliser towards inferring the full transitive closure. This included calculating the difference between the two matrices, which allowed us to track the number of actual discrepancies between the two. Listing 5.3 shows output that is written to the console of our IDE if we run the trial just described.

Listing 5.3: Neural symbolic inference at work.

```
Epoch: 1
```

```
Starting adjacency matrix has 299 entries
Updated adjacency matrix has 938 entries
Number of entries newly inferred: 639

Evaluating inference performance of neural inference model ...
Target transitive closure adj mat nonzero entries: 1175
Neural inference transitive closure adj mat nonzero entries: 938
Number of discrepancies between entries: 237

Epoch: 2

Starting adjacency matrix has 938 entries
Updated adjacency matrix has 1174 entries
Number of entries newly inferred: 236

Evaluating inference performance of neural inference model ...
Target transitive closure adj mat nonzero entries: 1175
Neural inference transitive closure adj mat nonzero entries: 1174
Number of discrepancies between entries: 1

Epoch: 3

Starting adjacency matrix has 1174 entries
Updated adjacency matrix has 1175 entries
Number of entries newly inferred: 1

Evaluating inference performance of neural inference model ...
Target transitive closure adj mat nonzero entries: 1175
Neural inference transitive closure adj mat nonzero entries: 1175
Number of discrepancies between entries: 0
```

We can see in Listing 5.3 that the inference performed by the Generaliser is indeed incremental, and tied to the epochs of training. We can also see that the full transitive closure is correctly inferred, as indicated by the 0 count of discrepancies at the end of epoch 3. We also observe that it takes only 3 epochs to infer the full transitive closure. But this speed of inference is somewhat an artefact of the special (simulated) conditions under which we are calling the Generaliser.

Crossing the subsymbolic/symbolic divide differentiably

Recall from Figure 5.1 that the architecture of our Combiner model contains within it what we call the subsymbolic/symbolic divide. This is the point at which the Combiner transitions from the subsymbolic world of the Classifier to the symbolic world of the Generaliser. The outputs of the Classifier (vectors of logits) are converted into inputs for the Generaliser (binary one-hot vectors).

In the investigation into our concept of 'neural symbolic inference' described above, we used a non-differentiable implementation of this conversion process. That was sufficient for our use case. But the architecture of the Combiner (a subsymbolic-symbolic neural network), with its subsymbolic/symbolic divide, begs the question: can the divide be crossed differentiably? We examined this question and developed a technique for doing

so. So the answer is yes.

Here we describe our approach for crossing the subsymbolic/symbolic divide of our Combiner model. Recall that the Classifier module within our Combiner model is a multi-class, single-label classifier. The approach we describe reflects this particular context. Our approach for crossing the subsymbolic/symbolic divide of our Combiner model, expressed generically, for a single vector of logits, is as follows:

- we declare a tiny epsilon value very near 0 (`epsilon`);
- we find the maximum value in the vector of logits (`maxval`);
- we compute a threshold value for the vector, `threshval`, equal to `maxval` minus `epsilon`
- we instantiate a threshold activation function for the vector, passing it `threshval` as the threshold value, and 0 as the replacement value;
- we threshold the vector of logits; All values below `threshval` are turned to 0, and the `maxval` value survives the thresholding intact, because `threshval` is a tiny bit less than `maxval`;
- then we divide the thresholded vector by `maxval` to convert the surviving nonzero value, `maxval`, to 1.

Using toy scenarios in our IDE, we confirmed that this procedure is indeed differentiable (with respect to PyTorch, at least). Gradients flow across the divide, from the symbolic module to the subsymbolic module and all of its parameters.

By contributing an approach for crossing the subsymbolic/symbolic divide of the Combiner model differentiably, we make end-to-end learning possible, within our Combiner model, and within the context of subsymbolic-symbolic neural network architectures generally. But it may not be immediately clear how such a capability might be exploited. Use cases and application tasks for such a setting may not leap immediately to mind. Yet, perhaps knowing that the capability for end-to-end learning exists may liberate imaginations enough for use case ideas to emerge over time.

Summary Before closing this section, Section 5.4.4, on neural symbolic inference, we briefly summarise what was discussed. We introduced the concept of neural symbolic inference in which a symbolic module within a neural network performs logical inference in the rhythm of a forward-pass of a neural network training cycle. We explained how we adapted an instance of the Generaliser component of our Combiner model to infer the transitive closure of a class hierarchy, incrementally, within the forward-pass flow of a standard training cycle. We demonstrated that the transitive closure inference capabilities of our Generaliser are robust by showing evidence that it is capable

of correctly inferring the transitive closure the class hierarchy of our VRD-World OWL ontology. We described the OCHAM tool that we created to manage our way around the current limitations of our tensor representation of an OWL-based knowledge graph. And we discussed the issue of crossing the subsymbolic-symbolic divide in our Combiner model, and described an approach for crossing that divide differentiably, in order to open possibilities for end-to-end learning.

5.4.5 Tensor knowledge graph embeddings

Recall that in Section 2.5, in our discussion of work related to thread three of our research (tensor knowledge graphs), we touched on the topic of matrix factorisation knowledge graph embedding models. We revisit that discussion here. We examine the strong affinity that exists between tensor knowledge graphs, the tensor knowledge graph reasoning engine, and matrix factorisation knowledge graph embedding models. We also specify a matrix factorisation for our tensor knowledge graph.

The knowledge graph embedding model that is the basis for most of the matrix factorisation knowledge graph embedding models discussed in Rossi et al. (2021) is called RESCAL (Nickel et al., 2011). We describe this model using the original RESCAL notation. RESCAL represents a knowledge graph in one 3D binary tensor X, of size $T \times n \times n$, where n is the number of individuals in the graph, and T is the number of relations (or predicates) used to link individuals. Let X_k denote the kth $n \times n$ relation encoded within tensor X (*i.e.*, channel k of X). The RESCAL factorisation factorises each $n \times n$ relation X_k individually, as follows:

$$X_k \approx A R_k A^T \quad \text{for } k = 1, \ldots, T, \tag{5.3}$$

where A is an $(n \times r)$ matrix containing the embeddings of the n individuals (in its rows), and R_k is an $r \times r$ matrix containing embeddings that model relations between individuals with respect to the kth relation (predicate). One performs the factorisation in order to obtain the embeddings, which can then be used in downstream applications.

Tensor X of the RESCAL model corresponds exactly to Tensor C of our tensor representation for OWL-based knowledge graphs, whose size we specified (in our notation) as $P \times N \times N$. This affinity between the RESCAL model and our tensor knowledge graph prompted us to derive a RESCAL-inspired factorisation for our tensor knowledge graph. We specify our tensor knowledge graph factorisation in the paragraphs below. We express our specification using a notation that is loosely based on the original RESCAL notation, but we have adapted it for our needs.

We let $X_k^{(A)}$ denote the counterpart of RESCAL matrix X_k for our Tensor A; $X_k^{(B)}$ denote the counterpart of RESCAL matrix X_k for our Tensor B; etc.. We let M denote the counterpart of RESCAL matrix A, the matrix that holds embeddings of entities. To

cope with the fact that our tensor knowledge graph has five tensors, not one, we use a superscript $(*)$, as in $M^{(*)}$, to distinguish between M matrices that are associated with our different tensors. Thus, $M^{(A)}$ is the counterpart of RESCAL matrix A for our Tensor A; $M^{(B)}$ is the counterpart of RESCAL matrix A for our Tensor B; etc.. Similarly, we use R_k as the basis for denoting relation matrices that contain relation embeddings, but we use superscripts to distinguish between the relation matrices for our five different tensors. Thus, $R_k^{(A)}$ denotes the kth relation matrix with respect to our Tensor A, and $R_k^{(B)}$ denotes the kth relation matrix with respect to our Tensor B, etc..

As a last note before we specify our RESCAL-inspired tensor knowledge graph factorisation, recall that tensor X in the RESCAL factorisation is $T \times n \times n$. In other words, each relation matrix X_k is square. The RESCAL factorisation, as specified, relies on X_k being square. Our tensor knowledge graph has five tensors, but only three of them have square channels. The other two have rectangular channels. Thus, for three of our five tensors, we use a RESCAL factorisation. For the other two tensors, we derived factorisations that are variations of a RESCAL factorisation. Finally, as a reminder, we let C denote the number of classes, P the number user-defined poperties (or predicates), and N the number of individuals.

Factorisation for Tensor A Tensor A of our tensor knowledge graph can be factorised using a pure RESCAL factorisation, as follows:

$$X_k^{(A)} \approx M^{(A)} R_k^{(A)} M^{(A)^T}, \tag{5.4}$$

$$(C \times C) = (C \times r^{(A)})(r^{(A)} \times r^{(A)})(r^{(A)} \times C)$$

Factorisation for Tensor B Tensor B of our tensor knowledge graph can be factorised using a pure RESCAL factorisation, as follows:

$$X_k^{(B)} \approx M^{(B)} R_k^{(B)} M^{(B)^T} \tag{5.5}$$

$$(P \times P) = (P \times r^{(B)})(r^{(B)} \times r^{(B)})(r^{(B)} \times P)$$

Factorisation for Tensor C Tensor C of our tensor knowledge graph can be factorised using a pure RESCAL factorisation, as follows:

$$X_k^{(C)} \approx M^{(C)} R_k^{(C)} M^{(C)^T} \tag{5.6}$$

$$(N \times N) = (N \times r^{(C)})(r^{(C)} \times r^{(C)})(r^{(C)} \times N)$$

Factorisation for Tensor D Tensor D of our tensor knowledge graph has rectangular channels, not square channels, so we cannot use a pure RESCAL factorisation here. We reuse matrix $M^{(A)}$ from the factorisation of Tensor A, and the matrix $M^{(C)}$ from the factorisation of Tensor C.

$$X_k^{(D)} \approx M^{(C)} R_k^{(D)} M^{(A)^T} \tag{5.7}$$
$$(N \times C) = (N \times r^{(C)})(r^{(C)} \times r^{(A)})(r^{(A)} \times C)$$

Factorisation for Tensor E Tensor E of our tensor knowledge graph has rectangular channels, not square channels, so we cannot use a pure RESCAL factorisation here. We reuse matrix $M^{(A)}$ from the factorisation of Tensor A, and the matrix $M^{(B)}$ from the factorisation of Tensor B.

$$X_k^{(E)} \approx M^{(B)} R_k^{(E)} M^{(A)^T} \tag{5.8}$$
$$(P \times C) = (P \times r^{(B)})(r^{(B)} \times r^{(A)})(r^{(A)} \times C)$$

Our RESCAL-inspired tensor knowledge graph factorisation, specified above, is intended to show one way in which tensor knowledge graphs can be usefully applied with respect to the topic of knowledge graph embeddings. The structural decomposition we outline here may be of interest to researchers in the embeddings community who are seeking knowledge graph embedding models that do a better job of capturing the rich expressiveness and logical relationships of OWL-based knowledge graphs. It is possible that researchers may find that our tensor representation of OWL-based knowledge graphs, and the decomposition guidance we provide here, help to generate knowledge graph embeddings that are more effective in downstream applications.

But it is not just the tensor knowledge graph itself (as a structure) that has strong potential for applications with respect to knowledge graph embedding models. The tensor knowledge graph reasoning engine does as well. With the tensor knowledge graph reasoning engine at hand, researchers exploring knowledge graph embedding models will have the option to generate embeddings from the *asserted* tensor knowledge graph, or from the *materialised* tensor knowledge graph, or both, for compare and contrast purposes. d'Amato et al. (2023) discuss the intuition that fully materialised OWL-based knowledge graphs, where everything implicit has been made explicit, contain more embeddable knowledge, and should therefore lead to embedding spaces that better reflect the totality of symbolic domain knowledge entailed by an OWL-based knowledge graph. They call for extensive research to study the potential of materialised knowledge graphs in knowledge graph embeddings, so that guidance with respect to whether, when and why to use knowledge graph materialisation for knowledge graph embeddings can evolve and mature. Our tensor knowledge graph and tensor knowledge graph reasoning engine are well placed to not just participate in such research, but to facilitate it.

5.5 Future work

Here we discuss ideas for planned and potential future work associated with tensor knowledge graphs and tensor knowledge graph reasoning. First we discuss the challenge of overcoming the key limitation of our tensor knowledge graph representation, which is its current inability to represent OWL class descriptions. Then we discuss forward chaining, a key feature we intend to implement. Next we describe the target use case for our concept of a tensor knowledge graph reasoning engine. Following this, we introduce the idea of knowledge graph comparison tools, and argue that tensor knowledge graphs provide good opportunities for building such services. Finally, we speculate about potential applications of the tensor knowledge graph reasoning engine in the field of AI planning.

5.5.1 Handling OWL class descriptions

Recall that our current tensor knowledge graph representation supports only named OWL classes with no description, and named OWL classes with the most simple of descriptions, such as

```
\texttt{(classA rdfs:subClassOf classB)} and
\texttt{(classC owl:equivalentClass classD)}.
```

Much of OWL's expressiveness and inference capability relies upon being able to describe classes by describing the characteristics that individuals must satisfy in order to be members of the class. In other words, an OWL class description describes a set: a set of individuals that possess a certain characteristic, or some mix of characteristics. The set may be empty (*i.e.*, no individuals in the knowledge graph currently satisfy the description of the class). If new data enters the knowledge graph, it may be that one or more individuals now do satisfy the class description and, if so, OWL reasoning will identify them and infer them to be members of the class. It will materialise what it infers with `rdf:type` triples, such as `(jane rdf:type Teacher)`. We think our tensor knowledge graph has potential to emulate this type of OWL reasoning, at least to some extent.

But an OWL class description can equate to more than just the *specification* of a set; the description may also have semantics which imply *how* the set is realised. That is, the description may also make reference to logical operations. To handle OWL class descriptions properly, our tensor knowledge graph needs to cope with both of these aspects. One idea for how to approach the problem is to develop techniques for parsing an OWL class description and for representing that description within the tensor knowledge graph as a kind of *class execution plan*. We illustrate this idea with a simple example.

Suppose OWL class C_k is described as being the logical union of classes C_i and C_j: *i.e.*, $C_k \equiv C_i \cup C_j$. The tensor knowledge graph has an $N \times C$ binary matrix that encodes the heterogeneous relation `rdf:type`, which maps individuals to classes. That matrix lives in channel 0 of Tensor D of the tensor knowledge graph. Within that $N \times C$ binary matrix for relation `rdf:type`, the individuals belonging to class C_i are identified by 1s in the column for class C_i. And the individuals belonging to class C_j are identified by 1s in the column for class C_j. To infer the members of class C_k, we need to logically union the columns for classes C_i and C_j, and then union the result into the column for class C_k. If we let i denote the column for class C_i, etc., then a conceptual tensor knowledge graph *class execution plan* for representing the OWL description of class C_k would express the following (in some manner, yet to be determined):

$$D[0, :, k] \leftarrow \Big(D[0, :, i] \cup D[0, :, j]\Big) \cup D[0, :, k]$$

5.5.2 Forward chaining

Our 'research and development' tensor knowledge graph reasoning engine currently performs only one reasoning pass over a tensor knowledge graph. We cannot expect a single reasoning pass to deliver 'complete' logical inference because the inference of one fact can trigger the inference of another. Therefore, one objective is to implement some form of *forward chaining* (Russell & Norvig, 2022) We intend to wrap our reasoning pass within a loop, and iterate over the loop until nothing new gets inferred, at which point we can stop.

5.5.3 Fast symbolic OWL reasoning for neurosymbolic systems

Our target use case for the tensor knowledge graph reasoning engine is to have it be used in the same way we used GraphDB ("Ontotext GraphDB", 2023) in thread two of our research: as a real-time symbolic OWL reasoning engine in our neurosymbolic systems for detecting visual relationships in images. Our vision for the tensor knowledge graph reasoning engine within this target setting is as follows. At instantiation, the engine consumes an OWL ontology file that contains an OWL ontology only, *i.e.*, no data triples. This is directly akin to loading a logic program into some reasoning engine.

To exercise the reasoning capability of the engine, one interacts with it in a particular way. An engine interaction involves putting to the engine some discrete tiny world of facts (some small collection of related triples) over which to reason. This is akin to presenting a logic program with some data. The reasoning the engine performs over the tiny world of the current engine interaction is restricted to that tiny world. That is, all evidence of the previous engine interaction, *i.e.*, the triples inserted, and any triples

inferred, will have been removed. We foresee the tensor knowledge graph reasoning engine managing this clear down of a previous interaction itself, automatically, as the first step of handling a new interaction.

An API of some kind (to be determined) enables the engine to return results pertaining to the most recent engine interaction, according to user needs. These needs likely include `(i)` the triples that were inferred by the most recent engine interaction, `(ii)` the triples that were asserted as part of the most recent engine interaction, `(iii)` all triples associated with the most recent engine interaction (asserted and inferred), or `(iv)` logical inconsistencies that have arisen, if any. Were our vision for the target use case of the tensor knowledge graph reasoning engine to be realised, it would be possible to repeat thread two of our research into leveraging OWL-based knowledge graphs and symbolic OWL reasoning to guide neural, subsymbolic learning in neurosymbolic systems for detecting visual relationships in images, as described in Chapter 4.

Our current 'research and development' tensor knowledge graph reasoning engine, however, does not fit our vision for our target use case. Currently, the engine permits only the one-time loading of an OWL ontology file, when the engine is instantiated. This file must contain an OWL ontology, plus any and all of the data triples with which one intends to work. In other words, a complete knowledge graph must be contained in one ontology file, and it can be loaded into the engine just once. To support our target use case, our 'research and development' tensor knowledge graph reasoning engine faces significant refactoring.

5.5.4 Knowledge graph comparison services

Another use case (application) we foresee for the tensor knowledge graph reasoning engine is to provide OWL-based knowledge graph comparison services. The services we imagine here (for now, at least) would not involve OWL reasoning in any way, however. The comparison service we foresee presumes identical knowledge graph structure (via use of a common OWL ontology) and focuses on content differences only.

The observation that prompted this application idea is that once an OWL-based knowledge graph has been represented as a tensor knowledge graph (with its five 3D tensor components), comparing that tensor knowledge graph with another would be straightforward (assuming that structurally the two are identical). Only the 'difference' between the five pairs of counterpart 3D tensors constituting the two tensor knowledge graphs would need be computed. The direction of the differences would be obtainable too, *i.e.*, triple X is present in graph G_1 but not in graph G_2, and triple Y is not present in graph G_1 but is present in graph G_2. In other words, the comparison services could operate in a manner analogous to the Unix command-line 'diff' utility for comparing two text files, and produce analogous output. Think 'knowledge graph diff' utility.

5.5.5 AI planning

Here we discuss a potential application area for the tensor knowledge graph reasoning engine that is more speculative: facilitating AI planning research that uses OWL ontologies and OWL reasoning. John and Koopmann (2024) use OWL-DL ontologies for AI planning, where 'DL' is a reference to Description logic. As they explain, AI planning involves deciding sequences of actions for achieving specific goals, and is important for enabling agents to behave autonomously. If OWL ontologies and OWL reasoning have a place in AI planning, then the tensor knowledge graph reasoning engine should be well-placed to participate.

We have wondered about applications of OWL ontologies and OWL reasoning in AI planning ourselves. Our intuition is based upon the observation that there is affinity between the notion of 'sequences of actions' and the property patterns that can be expressed using OWL *property chains*.

To illustrate, suppose an OWL ontology declares a simple property chain that says: if a chain of triples exists that involves property `brotherOf` followed by property `fatherOf`, then infer property `uncleOf`. An instance of this property chain, and the inference associated with it, is seen here:

```
since
    (john brotherOf robert) and (robert fatherOf jane)
infer
    (john uncleOf jane).
```

An important feature of OWL property chains is that they can be of any length, and they can be composed. For example, here we show a generic property chain of length 3 that is a composition of two property chains of length 2. This chain construction permits up to three new triples to be inferred, depending on data in the graph:

```
since
    (A p1 B) and (B p2 C) and (C p3 D)
infer
    (A p4 C), (B p5 D), and (A p6 D).
```

One can imagine large property chains, trees of related chains, forests of chain trees, etc.. It may be that the triples inferred by OWL reasoning, when it detects instances of property chains in data, can be used as information that feeds into planning decisions, and/or as direct signals of actions to be taken.

It appears feasible for the tensor knowledge graph reasoning engine to one day support the inference associated with OWL property chains. Property chains express relational composition. As we have seen, the engine implements relational composition using

matrix multiplication. So property chain inference should be fast. Hence, if OWL property chains do prove useful in modelling AI planning problems, then the tensor knowledge graph reasoning engine will be even better placed to participate.

5.6 Summary

This chapter covered thread three of our research into combining subsymbolic learning with symbolic OWL reasoning. Thread three centres on our notion of a tensor knowledge graph. First we focused on concepts: tensor knowledge graphs and tensor knowledge graph reasoning techniques, based on relational mathematics. We explained the idea of a tensor knowledge graph, and then we detailed a specification for our tensor knowledge graph representation of an OWL-based knowledge graph. We provided thorough coverage of our approach to tensor knowledge graph reasoning, where the goal is to emulate aspects of OWL reasoning (within limitations). Then our focus shifted to applications of tensor knowledge graph concepts. We discussed several categories of these, some of which involved combining tensor knowledge graph encoded symbolic knowledge, and tensor knowledge graph reasoning techniques, within novel neurosymbolic system architectures. Finally, we discussed several categories of planned and potential future work that lie ahead in relation to tensor knowledge graphs.

As part of our coverage of tensor knowledge graph reasoning, we did the following:

- we presented our matrix-based implementations of three core binary relational operations that we leverage as re-usable reasoning building blocks throughout our tensor knowledeg graph reasoning techniques;
- to illustrate tensor knowledge graph reasoning in practice, we gave detailed accounts of how our tensor knowledge graph reasoning engine emulates the inference semantics of OWL properties `owl:inverseOf` and `rdfs:domain`;
- in connection with property `rdfs:domain`, we highlighted that aspects of our tensor knowledge graph reasoning techniques demonstrate both efficiency and scalability
- we summarised the RDFS and OWL inference semantics that our tensor knowledge graph; reasoning engine currently supports, and we pointed to the fact that this support is already equivalent to what other sources in the Semantic Web community refer to as *RDFS-Plus*, a recognised subset of OWL that includes the most useful and commonly used features of OWL
- we did a deep dive into two matrix-based transitive closure algorithms, because efficient transitivity inference is important for ensuring the viability of the tensor knowledge graph reasoning engine in practical settings;

- we described early-stopping conditions that we identified for use with one of the two transitive closure algorithms, and we showed that these stopping conditions make that algorithm significantly faster, and without compromising the logical soundness of the inference;
- and we compared the runtime efficiency of the two transitive closure algorithms, and showed that, counter-intuitively, the algorithm with greater computational complexity is actually much faster than the algorithm with lower complexity.

As part of our discussion of applications of tensor knowledge graph concepts, we did the following:

- We introduced our notion of a subsymbolic-symbolic neural network architecture, and described an instance of that architecture that we refer to as the Combiner, which wraps a subsymbolic Classifier module with a symbolic Generaliser module that generalises the classes predicted by the Classifier. We discussed an investigation that involved the *injection* of the symbolic knowledge of a class hierarchy, encoded by the tensor knowledge graph reasoning engine in a binary matrix, into the weight matrix of a Generaliser, enabling it to find all of the parent classes of a base class via knowledge retrieval from its injected knowledge base, automatically, via the conventional matrix algebra of a neural network forward-pass. Then we discussed an extension of that investigation in which the objective was to have (a subsymbolic incarnation of) the Combiner *learn* the symbolic knowledge of a class hierarchy for itself, rather than injecting the knowledge.
- We introduced our notion of 'neural symbolic inference', where the symbolic component of our proposed subsymbolic-symbolic neural network architecture performs symbolic logical inference. We illustrated this notion by describing an investigation in which the Generaliser module of our Combiner model (an instance of a subsymbolic-symbolic architecture) is equipped to infer the transitive closure of a class hierarchy, incrementally, within the rhythm of a neural network forward-pass. The inference algorithm used by this Generaliser was transferred from our tensor knowledge graph reasoning engine, and adapted to fit the forward-pass flow of a neural network. We described how we used the class hierarchy of our VRD-World ontology (part of NeSy4VRD) to verify that the transitive closure inferred by the Generaliser matched exactly the transitive closure of the class hierarchy as inferred by OWL reasoning. We ended our discussion of neural symbolic inference by focusing on the subsymbolic/symbolic divide that exists within the subsymbolic-symbolic network architecture. We presented an approach to crossing this divide differentiably—a capability which opens opportunities for end-to-end learning (and perhaps reasoning) within the setting of instances of a subsymbolic-symbolic network architecture, such as our Combiner model.

- We argued that there are compelling opportunities for our tensor knowledge graph, and our tensor knowledge graph reasoning engine, to participate in research associated with knowledge graph embeddings and the development of knowledge graph embedding models. We contributed a RESCAL-inspired specification of a factorisation of our tensor knowledge graph model in order to facilitate such research.

With respect to future work, we discussed the priorities of *(i)* extending tensor knowledge graph capabilities so they can handle (and reason with) arbitrary OWL class descriptions (rather than just class names, as at present), and *(ii)* extending tensor knowledge graph reasoning capabilities to include a form of forward chaining so that it can deliver inference results that are not just logically sound, but logically complete as well. Then we described the target use case for our vision of a tensor knowledge graph reasoning engine, which is to serve as a symbolic reasoning engine in neurosymbolic systems, and where an OWL ontology is leveraged in a manner akin to a logic program, for reasoning over discrete, tiny worlds, in rapid succession. Next we described our vision of the tensor knowledge graph reasoning engine evolving to the point where it can act like a knowledge graph 'diff' utility, to provide knowledge graph comparison services. Finally, we closed with brief speculative thoughts on potential applications for the tensor knowledge graph reasoning engine within the research area of AI planning, where the processing efficiency of our engine may possibly be leveraged to great effect, particularly if the inference required involves the processing of long, dense and complex OWL property chains.

Overall, this chapter has shown that it is feasible to emulate many aspects of symbolic OWL reasoning using a binary tensor representation of an OWL-based knowledge graph together with logical inference techniques that rely on binary matrix manipulation. The building blocks of our matrix-based OWL inference techniques are direct counterparts of binary relational operations; they reflect known default approaches to implementing those relational operations whenever a binary relation is represented as a binary matrix rather than a directed graph. Our approach is not heuristic; it is well-grounded in mathematical theory. And thus our concept of an OWL-based tensor knowledge graph reasoning engine has legitimacy that deserves to be recognised.

Lastly, recall our central research questions: *(i)* how can we combine subsymbolic learning with symbolic OWL reasoning, and *(ii)* what are the effects or benefits of doing so? This chapter has addressed these questions from the perspective of our notion of OWL-based tensor knowledge graphs. The chapter establishes that there exists a rich variety of real and potential application opportunities for tensor knowledge graphs, and for our tensor knowledge graph reasoning engine, in neurosymbolic systems research. This chapter concludes the third and final thread of our research into neurosymbolic learning and reasoning with OWL-based knowledge graphs.

Chapter 6

Summary

Our presentation of our research into neurosymbolic learning and reasoning with OWL-based knowledge graphs is now complete. In this chapter, we summarise the ground covered, and recall our contributions. As we do so, we reflect upon our central research questions: *(i)* how can we combine subsymbolic learning with symbolic OWL reasoning, and *(ii)* what are the effects or benefits of doing so?

Recall that our research has three threads. Thread one, presented in Chapter 3, concerned NeSy4VRD, the 'image dataset with OWL ontology' resource that we created to enable thread two. Thread two, presented in Chapter 4, examined ways of leveraging OWL-based knowledge graphs, and OWL reasoning, for the computer vision task of detecting visual relationships in images. Thread three, presented in Chapter 5, introduced tensor knowledge graphs and tensor knowledge graph reasoning, and discussed applications of these concepts in neurosymbolic systems. Here, we review each of these threads, in turn, before ending with brief closing remarks.

6.1 Thread one

In thread one of our research, we presented NeSy4VRD. NeSy4VRD is a resource that supports research into neurosymbolic learning and reasoning with OWL-based knowledge graphs. It is a neurosymbolic systems research enabler, within the context of the computer vision task of detecting visual relationships in images. The NeSy4VRD dataset restores public access to the images of the VRD dataset (Lu et al., 2016), and combines these with quality-improved NeSy4VRD annotated visual relationships. Uniquely, NeSy4VRD accompanies its dataset with a well-aligned, custom-designed, common sense, companion OWL ontology, called VRD-World. The alignment between VRD-World and the NeSy4VRD dataset is tight—tight enough that the results (infer-

ences) of OWL reasoning are directly pertinent to the underlying subsymbolic learning task of detecting visual relationships in images.

The baseline NeSy4VRD annotations, and the companion OWL ontology, VRD-World, can be used *as is*, or they can be adjusted and extended. NeSy4VRD is designed for extensibility. It invites researchers to tailor it to their needs, and to extend it, and to share their extensions. It does this by providing comprehensive software infrastructure in support of extensibility. This support spans *(i)* extensive functionality for analysing the annotated visual relationships, and hence the images, of the NeSy4VRD dataset, *(ii)* a custom-designed text-based protocol for specifying customisations and extensions to annotated visual relationships, declaratively, in text files, and *(iii)* a configurable, multi-step workflow for applying annotation customisations and extensions in an automated, repeatable fashion.

NeSy4VRD also introduces and proposes a novel process model for collaborative and decentralised enrichment of annotated datasets. We call this process model *distributed annotation enhancement* (DAE). The vision for DAE with respect to NeSy4VRD is for NeSy4VRD annotated visual relationships (and, by extension, the VRD-World OWL ontology) to be progressively extended and enriched, over time, in a decentralised fashion, through the independent actions of researchers pursuing their independent research interests. Our DAE process model for NeSy4VRD supports the sharing of extensions. It also supports the utilisation of shared extensions, by any researcher, by providing the means to compose the baseline NeSy4VRD annotated visual relationships with any mix of shared extensions.

We contributed NeSy4VRD to the computer vision and neurosymbolic AI communities, as announced in Herron, Jiménez-Ruiz, Tarroni, and Weyde (2023), to enable more research similar to ours. NeSy4VRD enables neurosymbolic research with OWL-based knowledge graphs by lowering the barriers to entry for conducting such research.

6.2 Thread two

In thread two of our research, we used NeSy4VRD to explore ways of combining subsymbolic learning with symbolic OWL reasoning in neurosymbolic systems designed for detecting visual relationships in images. We presented two investigations that used different strategies for addressing this problem. And we described a further strategy, involving Datalog-like rules that extend OWL's reasoning capability, that we left for future work.

Investigation 1 In investigation 1 of thread two, we used OWL-based knowledge graph reasoning to augment the annotated scene graphs of images, thereby enriching the

supervision they provide during neural network training. We observed that this strategy for using OWL reasoning to guide neural, subsymbolic learning had a general 'positive' character to it, in the sense that the strategy is designed to help neural networks learn what *to* predict, rather than what *not to* predict.

In investigation 1, we focused on exploring the category of OWL reasoning we refer to as 'link inference', where new relations are inferred between individuals, and are materialised as new triples in a knowledge graph (and new visual relationships in an image scene graph). We examined five sub-categories of 'link inference' capability: the OWL inference semantics associated with symmetry, inverses, transitivity, equivalence of properties, and sub-property relationships. We examined using multiple different versions of our OWL ontology, VRD-World, each possessing a different mix of these five 'link inference' capabilities, to govern OWL reasoning in the augmentation of the image scene graphs. And when we examined the effects of such OWL reasoning, as measured by recall@N, we saw lively, almost musical, responsiveness from the neural network to the stimulus induced by OWL reasoning.

Suppose, for a moment, we liken the five sub-categories of 'link inference' capability to *knobs* on our VRD-World OWL ontology—knobs we can turn, on and off, to control what OWL reasoning can and cannot do. We turned the knobs on our ontology, and we saw OWL reasoning make the neural network's recall@N scores dance to the ontology's tune. We saw precision sometimes double, and more. We saw the ontology determine, and generally increase, the network's speed of learning. If the symmetry knob was 'on', OWL reasoning induced the network to predict more symmetric pairs. If the inverses knob was 'on', OWL reasoning induced the network to predict more inverse pairs. Our evidence shows the ontology matters: it controlled the OWL reasoning that augmented the image scene graphs, and the neural network responded to the adjustments in the supervision it received.

Investigation 2 In investigation 2 of thread two, we explored using an OWL-based knowledge graph tool, hosting our VRD-World OWL ontology, as a binary classifier—a symbolic reasoning binary classifier. Its role was to classify predicted visual relationships (presented as tiny worlds) as being either semantically valid or semantically invalid. The yardstick was the collection of common sense domains and ranges declared in the VRD-World ontology for its user-defined properties—the counterparts of NeSy4VRD visual relationship predicates. We explained that the mechanism underlying the symbolic classifier relied upon exercising a combination of the categories of OWL reasoning we refer to as 'type inference' (for inferring class membership) and 'logical consistency inference' (for detecting logical inconsistencies).

We demonstrated using the symbolic reasoning binary classifier in three contexts: neural network training, neural network inference, and interpretation of experiment results.

In the neural network training setting, we demonstrated using the real-time symbolic reasoning binary classifier to help train a neural network (subsymbolic learning) classifier. We explained how we used the decisions of the symbolic classifier to apply a semantic loss penalty to the neural network under training, as a logical constraint, to help it learn the common sense semantics of the visual relationship predicates. We observed that this strategy for using OWL reasoning to guide neural, subsymbolic learning had a general 'negative' character to it, in the sense that the strategy is designed to help neural networks learn what *not to* predict, rather than what *to* predict. We saw that this strategy for leveraging OWL reasoning was highly effective, and significantly reduced (in relative terms) the frequency of semantically invalid predictions. But we also observed that the strategy induced fewer predictions overall, which registered as a small drop in recall@N. This outcome suggests there is an interesting side-effect of the semantic loss penalty (logical constraint) strategy at work, as if its effects are felt more widely than intended. Exploring how to narrow this spill-over, and focus the constraint more precisely, could be a subject of potential future work.

In the neural network inference setting, we demonstrated using the real-time symbolic reasoning binary classifier to identify predictions that were semantically invalid, so that these could be filtered-out, thereby enabling the neurosymbolic system to submit only semantically valid predictions of visual relationships for performance evaluation. We saw that this strategy for leveraging OWL reasoning in a neurosymbolic system was completely effective.

With respect to interpreting experiment results, we explained how the granular analysis of individual predicted visual relationships that we conducted, for both the neural network training and inference settings, would have been intractable were it not for the symbolic reasoning binary classifier. It did not just play a starring role in experiments, it made feasible evaluation of its effects.

Future work In addition to our two main investigations for thread two of our research, we also outlined a further investigation that we left for future work. The future investigation involves extending OWL reasoning capability with Datalog-like rules. The setting we described involved a visual relationship plausibility rule engine that would co-operate with an OWL-based knowledge graph (acting as a real-time symbolic reasoning engine) in the execution of Datalog-like rules for inferring plausible predictions. We introduced the notion of a semantic plausibility penalty. This notion turns the logical constraint from investigation 2 (just described) on its head. The semantic loss penalty from investigation 2 is a logical constraint designed to suppress bad predictions. The semantic plausibility penalty is a logical constraint designed to induce good predictions, especially when the network is not, on its own, showing an inclination to predict them. In fact, we may think of this idea as a *logical inducement* rather than *constraint*, and

speak of a *semantic plausibility prompting* rather than *penalty*. The lower the network's confidence in a prediction inferred (with the help of OWL reasoning) to be plausible, the greater the prompting to learn that prediction.

Conclusions The two investigations we presented for thread two of our research, and the third investigation we left for future work, all involve using our custom-designed, common sense OWL ontology, VRD-World, hosted in OWL-based knowledge graph tools, in the guise of symbolic (OWL) reasoning engines. All three investigations rely on using VRD-World as a counterpart of a logic program—a program that governs the inference performed by OWL reasoning within the symbolic engine. We demonstrated use of these engines in batch mode, and in real-time; and during neural network training, inference, and even for analysis of experiment results. This real-time symbolic reasoning engine setting for OWL-based knowledge graph technologies is under-explored in neurosymbolic systems. Our research helps to cast these technologies in a new light.

6.3 Thread three

In thread three of our research, we explored our central research questions from a perspective quite different to that of thread two. In thread three, we presented our notion of tensor knowledge graphs, explained our approach to tensor knowledge graph reasoning, and described applications of these concepts in neurosymbolic systems. We review each of these areas and recall what was discussed.

Tensor knowledge graph concepts In Section 1.2, we provided background to our research that focused on topics relating to the symbolic tradition of AI. We mentioned knowledge representation and reasoning (KRR). And we pointed out that, as a package, OWL, OWL ontologies, and OWL reasoning can be thought of as representing an instantiation of KRR.

The notion of a tensor knowledge graph explores one way of representing the symbolic knowledge of an OWL-based knowledge graph. And tensor knowledge graph reasoning explores techniques for reasoning over that knowledge that are tailored for the tensor knowledge graph representation of symbolic knowledge. Thus, a good way to understand our notions of tensor knowledge graphs, and of tensor knowledge graph reasoning, is to see them too as an instantiation of KRR. They are an instantiation of KRR that seeks to emulate aspects of the OWL-based instantiation of KRR. The underlying logic formalism is the same: the Description logic $\mathcal{SROIQ}$, and its Semantic Web counterpart, OWL.

We explained that OWL-based knowledge graphs, and tensor knowledge graphs, seek to represent the same symbolic knowledge, but in different ways. We observed that the

most general way to conceptualise the symbolic knowledge in question was to see it as collections of binary relations, which, by definition, are sets of ordered pairs. OWL and OWL ontologies represent these binary relations as directed graphs. Tensor knowledge graphs represent these same binary relations as binary matrices, without loss of information. OWL reasoning techniques exploit the directed graph representation of the binary relations of the symbolic knowledge. Tensor knowledge graph reasoning techniques exploit the binary matrix representation of the binary relations of the symbolic knowledge.

The two KRR instantiations are intertwined. The tensor knowledge graph-based instantiation seeks to emulate the OWL-based instantiation. We argued that the tensor knowledge graph approach to emulating OWL is not heuristic. We showed that our reasoning techniques are based upon basic binary relational operations that are grounded in relational mathematics, and especially in the body of mathematical theory at its core, called the *Calculus of Relations*. We showed that, even at these early stages of its development, within the current limitations of tensor knowledge graphs, tensor knowledge graph reasoning is already mature enough to emulate the equivalent of what sources in the Semantic Web community refer to as *RDFS-Plus* inference semantics.

We did a deep dive into matrix-based transitive closure algorithms used by our tensor knowledge graph reasoning engine. One of them, seemingly disregarded, likely due to its high complexity of $O(n^4)$, we could barely find references to, let alone a name. We embraced this algorithm and called it the Union-of-Powers. We studied its behaviour closely and found early-stopping conditions to make it more efficient. And we showed that, on current computing hardware, optimised as it is for tensor and matrix operations, the Union-of-Powers algorithm is orders of magnitude faster than the alternate $O(n^3)$ algorithm, highlighting that complexity analysis can be an unreliable guide as to which algorithms to prefer.

Tensor knowledge graph applications Once we had established the concepts of tensor knowledge graphs and tensor knowledge graph reasoning, we shifted the focus to applications of these concepts. We began by introducing the concept of a subsymbolic-symbolic neural network architecture, and presented our Combiner as an instance of this architecture. The Combiner wrapped a subsymbolic learning Classifier module with a symbolic reasoning Generaliser module. We then described a series of investigations based upon the Combiner architecture that explored different ways of combining subsymbolic learning with symbolic knowledge, and with symbolic reasoning, within the setting of a standard neural network forward-pass. These investigations relied upon just one aspect of the symbolic knowledge that a tensor knowledge graph encodes in order to represent an OWL ontology: the ontology's declared class hierarchy, and its transitive closure.

We explored injecting this encoded symbolic knowledge directly into the weight matrix of a symbolic Generaliser. And we showed that, once a base class predicted by a subsymbolic Classifier has been rendered symbolically, as a binary one-hot vector, the matrix algebra of a network forward-pass is all that is needed for the Generaliser to retrieve the knowledge of the correct parent classes from its injected knowledge base. We distinguished between injection of full knowledge (the transitive closure of an OWL class hierarchy) and injection of partial knowledge (the class hierarchy as declared in an OWL ontology). We explained how a Generaliser can handle both scenarios, and still always generalise correctly, to return the full set of parent classes. We also described an investigation into having a fully subsymbolic Combiner learn the a symbolic (binary matrix) encoding of a class hierarchy transitive closure.

Next we introduced our notion of 'neural symbolic inference', in which a symbolic Generaliser in a Combiner is equipped to do proper symbolic logical inference, within the rhythm of a neural network forward-pass. We described how we adapted a transitive closure inference algorithm used by our tensor knowledge graph reasoning engine for the setting of a neural network forward pass. And we showed that, so equipped, a Generaliser was able to fully infer the transitive closure of the VRD-World class hierarchy.

We also discussed crossing the subsymbolic/symbolic divide that exists in a subsymbolic-symbolic network architecture. We described an algorithm for crossing this divide differentiably, to open opportunities for end-to-end learning within the setting of a subsymbolic-symbolic architecture. It is not immediately clear what this means, or where this might lead. But it does create novel conditions in which to explore the subject of combining subsymbolic learning with symbolic reasoning.

Tensor knowledge graph future work We closed our discussions of tensor knowledge graphs (thread three of our research) by describing several categories of planned and potential future work. We outlined an approach to handling OWL class descriptions that relies upon a concept we named a *class execution plan*. We described how we foresee wrapping our current reasoning techniques in a loop, in order to implement a form of forward chaining, with the objective of delivering inference that is not just logically sound, but complete as well. We described the target use case for our notion of a tensor knowledge graph reasoning engine: a real-time symbolic (OWL) reasoning engine, governed by a custom OWL ontology acting as a logic program, and presented with a succession of tiny worlds over which to reason, for use in neurosymbolic systems. We discussed applications of tensor knowledge graphs in relation to certain knowledge graph embedding models. We highlighted the affinity that exists between our tensor knowledge graph and knowledge graph embedding models that take a matrix factorisation approach to generating embeddings. And we specified a RESCAL-inspired matrix

factorisation scheme for our tensor knowledge graph, to facilitate its use in knowledge graph embedding model research. We ended with a more speculative discussion of the possibility of finding applications for a tensor knowledge graph reasoning engine in AI planning problems.

6.4 Closing remarks

We have examined neurosymbolic learning and reasoning with OWL-based knowledge graphs from a variety of perspectives. And our research has yielded a variety of insights in this regard. One way to view our explorations of combining subsymbolic learning with symbolic OWL reasoning is through the lens of the depth of the neurosymbolic integration. In thread two, it was shallow. The influence of OWL reasoning was transmitted via training targets, and via the loss function (whether by a semantic loss penalty, or, in future, by a semantic plausibility prompting). In thread three, the integration was deeper. We injected encoded symbolic knowledge into symbolic modules of network architectures, and equipped symbolic modules of networks to perform logical inference on encoded symbolic knowledge, in the rhythm of a forward-pass.

In Herron et al. (2025), we pointed to the potential that OWL-based knowledge graphs, and OWL reasoning, have for contributing in neurosymbolic systems. Here we have demonstrated some of that potential, and pointed to more yet. Our research casts OWL-based knowledge graph technologies, and OWL reasoning, in a little-explored light—as components of symbolic reasoning engines in neurosymbolic systems. It introduces a new approach to thinking about OWL reasoning itself. And it explores new ways of thinking about what it means to combine subsymbolic learning with symbolic reasoning, in novel network architectures. Our research may expand perceptions in these multiple dimensions of neurosymbolic AI.

Bibliography

Abboud, R., Ceylan, I., Lukasiewicz, T., & Salvatori, T. (2020). BoxE: A Box Embedding Model for Knowledge Base Completion. *Advances in Neural Information Processing Systems*, *33*, 9649–9661. https://proceedings.neurips.cc/paper_files/paper/2020/file/6dbbe6abe5f14af882ff977fc3f35501-Paper.pdf

Allemang, D., Hendler, J., & Gandon, F. (2020). *Semantic Web for the Working Ontologist* (3rd ed.). ACM Books.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. F. (Eds.). (2007). *The Description Logic Handbook* (2nd ed.). Cambridge University Press.

Baader, F., Horrocks, I., Lutz, C., & Sattler, U. (2017). *An Introduction to Description Logic*. Cambridge University Press.

Badreddine, S., d'Avila Garcez, A., Serafini, L., & Spranger, M. (2022). Logic Tensor Networks. *Artificial Intelligence*, *303*. https://doi.org/10.1016/j.artint.2021.103649

Bang-Jensen, J., & Gutin, G. Z. (2009). *Digraphs - Theory, Algorithms and Applications* (2nd ed.). Springer.

Behnke, R., Berghammer, R., Meyer, E., & Schneider, P. (1998). RELVIEW - A System for Calculating With Relations and Relational Programming. *Fundamental Approaches to Software Engineering, 1st International Conference, FASE 1998, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 1998*, *1382*, 318–321. https://doi.org/10.1007/BFb0053599

Berghammer, R., & Neumann, F. (2005). RelView - An OBDD-Based Computer Algebra System for Relations. *Computer Algebra in Scientific Computing, 8th International Workshop, CASC 2005*, *3718*, 40–51. https://doi.org/10.1007/11555964%5C_4

Berghammer, R., & Schmidt, G. (1993). RELVIEW — A Computer System for the Manipulation of Relations. *3rd conference on Algebraic Methodology and Software Technology (AMAST'93)*, 403–404.

Berghammer, R., & Schmidt, G. (2007). Algebraic Visualization of Relations Using RelView. *Computer Algebra in Scientific Computing*, 58–72.

Berghammer, R., von Karger, B., & Ulke, C. (1996). Relation-Algebraic Analysis of Petri Nets with RELVIEW. *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS 1996*, *1055*, 49–69. https://doi.org/10.1007/3-540-61042-1%5C_38

Berghammer, R., & Winter, M. (2014). Gunther Schmidt's life as a mathematician and computer scientist. *J. Log. Algebraic Methods Program.*, *83*(2), 300–308. https://doi.org/10.1016/j.jlap.2014.02.015

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, *284*(5), 34–43. https://www.jstor.org/stable/pdf/26059207.pdf

Besold, T. R., d'Avila Garcez, A. S., Bader, S., Bowman, H., Domingos, P. M., Hitzler, P., Kühnberger, K.-U., Lamb, L., Lowd, D., Lima, P. M. V., de Penning, L., Pinkas, G., Poon, H., & Zaverucha, G. (2017). Neural-Symbolic Learning and Reasoning: A Survey and Interpretation. *CoRR*, *abs/1711.03902*. https://arxiv.org/abs/1711.03902

Bianchi, F., & Hitzler, P. (2019). On the Capabilities of Logic Tensor Networks for Deductive Reasoning. *Proceedings of the AAAI-MAKE*, *2350*.

Bordes, A., Usunier, N., García-Durán, A., Weston, J., & Yakhnenko, O. (2013). Translating Embeddings for Modeling Multi-relational Data. *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems*, 2787–2795. https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html

Brachman, R., & Levesque, H. (2004). *Knowledge Representation and Reasoning*. Morgan Kaufman.

Bratko, I. (2012). *Prolog Programming for Artificial Intelligence* (4th ed.). Addison Wesley.

Breit, A., Waltersdorfer, L., Ekaputra, F. J., Sabou, M., Ekelhart, A., Iana, A., Paulheim, H., Portisch, J., Revenko, A., Teije, A. t., & van Harmelen, F. (2023). Combining Machine Learning and Semantic Web: A Systematic Mapping Study. *ACM Computing Surveys*. https://doi.org/10.1145/3586163

Buffelli, D., & Tsamoura, E. (2023). Scalable Theory-Driven Regularization of Scene Graph Generation Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(6), 6850–6859. https://ojs.aaai.org/index.php/AAAI/article/view/25839

Chang, M., D'Aniello, G., Gaeta, M., Orciuoli, F., Sampson, D. G., & Simonelli, C. (2020). Building Ontology-Driven Tutoring Models for Intelligent Tutoring Systems Using Data Mining. *IEEE Access*, *8*, 48151–48162. https://doi.org/10.1109/ACCESS.2020.2979281

Chaudhri, V. K., Baru, C., Chittar, N., Dong, X. L., Genesereth, M., Hendler, J., Kalyanpur, A., Lenat, D. B., Sequeda, J., Vrandečić, D., & Wang, K. (2022). Knowledge graphs: Introduction, history, and perspectives. *AI Magazine*, *43*(1), 17–29. https://onlinelibrary.wiley.com/doi/abs/10.1002/aaai.12033

Chen, J., Geng, Y., et al. (2021). Low-Resource Learning with Knowledge Graphs: A Comprehensive Survey. *CoRR*. https://arxiv.org/abs/2112.10006

Chen, J., Hu, P., Jiménez-Ruiz, E., Holter, O. M., Antonyrajah, D., & Horrocks, I. (2021). OWL2Vec*: Embedding of OWL Ontologies. *Mach. Learn.*, *110*(7), 1813–1845. https://doi.org/10.1007/s10994-021-05997-6

Chen, M., Zhang, Y., Kou, X., Li, Y., & Zhang, Y. (2021). r-GAT: Relational Graph Attention Network for Multi-Relational Graphs. *CoRR*, *abs/2109.05922*. https://arxiv.org/abs/2109.05922

Chen, Z., Wang, Y., Zhao, B., Cheng, J., Zhao, X., & Duan, Z. (2020). Knowledge Graph Completion: A Review. *IEEE Access*, *8*, 192435–192456. https://doi.org/10.1109/ACCESS.2020.3030076

Chen, Z., Chen, J., et al. (2021). Zero-Shot Visual Question Answering using Knowledge Graph. *CoRR*. https://arxiv.org/abs/2107.05348

Corcoglioniti, F., Rospocher, M., Mostarda, M., & Amadori, M. (2015). Processing Billions of RDF Triples on a Single Machine using Streaming and Sorting. *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 368–375. https://doi.org/10.1145/2695664.2695720

Dai, Y., Wang, S., Xiong, N. N., & Guo, W. (2020). A Survey on Knowledge Graph Embedding: Approaches, Applications and Benchmarks. *Electronics*, *9*(5). https://doi.org/10.3390/electronics9050750

d'Amato, C. (2020). Machine Learning for the Semantic Web: Lessons learnt and next research directions. *Semantic Web*, *11*(1), 195–203. https://doi.org/10.3233/SW-200388

d'Amato, C., Mahon, L., Monnin, P., & Stamou, G. (2023). Machine Learning and Knowledge Graphs: Existing Gaps and Future Research Challenges. *Transactions on Graph Data and Knowledge (TGDK)*, *1*(1), 8:1–8:35. https://doi.org/10.4230/TGDK.1.1.8

Donadello, I., Dragoni, M., & Eccher, C. (2019). Persuasive Explanation of Reasoning Inferences on Dietary Data. *Proceedings of the 1st Workshop on Semantic Explainability, co-located with the 18th International Semantic*

Web Conference (ISWC 2019), *2465*, 46–61. https://ceur-ws.org/Vol-2465/semex%5C_paper2.pdf

Donadello, I., & Serafini, L. (2019). Compensating Supervision Incompleteness with Prior Knowledge in Semantic Image Interpretation. *IJCNN Hungary, July 14-19*, 1–8. https://doi.org/10.1109/IJCNN.2019.8852413

Ebrahimi, M., Eberhart, A., Bianchi, F., & Hitzler, P. (2021). Towards Bridging the Neuro-Symbolic Gap: Deep Deductive Reasoners. *Appl. Intell.*, *51*(9), 6326–6348.

Ebrahimi, M., Sarker, M. K., Bianchi, F., et al. (2021). Neuro-Symbolic Deductive Reasoning for Cross-Knowledge Graph Entailment. *Proceedings of the AAAI-MAKE*, *2846*. http://ceur-ws.org/Vol-2846/paper8.pdf

Ekaputra, F. J., Llugiqi, M., Sabou, M., Ekelhart, A., Paulheim, H., Breit, A., Revenko, A., Waltersdorfer, L., Farfar, K. E., & Auer, S. (2023). Describing and Organizing Semantic Web and Machine Learning Systems in the SWeMLS-KG. *Proceedings of the Extended Semantic Web Conference (ESWC)*, *13870*, 372–389. https://doi.org/10.1007/978-3-031-33455-9%5C_22

Fu, P., Zhang, Y., Wang, H., Qiu, W., & Zhao, J. (2023). Revisiting the Knowledge Injection Frameworks. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 10983–10997. https://aclanthology.org/2023.emnlp-main.677

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012). *Answer Set Solving in Practice*. Springer.

Gelfond, M., & Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. *Proceedings of International Logic Programming Conference and Symposium*, 1070–1080. http://www.cs.utexas.edu/users/ai-lab?gel88

Geng, Y., Chen, J., Ye, Z., Yuan, Z., Zhang, W., & Chen, H. (2021). Explainable Zero-shot Learning via Attentive Graph Convolutional Network and Knowledge Graphs. *Semantic Web*, *12*(5), 741–765. https://doi.org/10.3233/SW-210435

Giunchiglia, E., Stoian, M. C., Khan, S., Cuzzolin, F., & Lukasiewicz, T. (2023). ROAD-R: The Autonomous Driving Dataset with Logical Requirements. *Machine Learning*. https://doi.org/10.1007/s10994-023-06322-z

Givant, S. (2017). *Introduction to Relation Algebras* (Vol. 1). Springer.

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: An OWL 2 Reasoner. *J. Autom. Reason.*, *53*(3), 245–269. https://doi.org/10.1007/s10817-014-9305-1

Green, T. J., Huang, S. S., Loo, B. T., & Zhou, W. (2013). Datalog and Recursive Query Processing. *Foundations and Trends in Databases*, *5*(2), 105–195. https://doi.org/10.1561/1900000017

Gross, J. L., Yellen, J., & Anderson, M. (2019). *Graph Theory and its Applications* (3rd ed.). CRC Press.

Harris, S., & Seaborne, A. (2013). SPARQL 1.1 Query Language. *W3C Recommendataion, W3C*. https://www.w3.org/TR/sparql11-query/

Hays, P. J., & Patel-Schneider, P. F. (2014). *RDF 1.1 Semantics* (W3C Recommendation). World Wide Web Consortium. https://www.w3.org/TR/rdf11-mt/

Herron, D., Jiménez-Ruiz, E., Tarroni, G., & Weyde, T. (2023). NeSy4VRD: A Multifaceted Resource for Neurosymbolic AI Research using Knowledge Graphs in Visual Relationship Detection. *CoRR*. http://arxiv.org/abs/2305.13258

Herron, D., Jiménez-Ruiz, E., & Weyde, T. (2023). On the Benefits of OWL-based Knowledge Graphs for Neural-Symbolic Systems. *NeSy 2023: 17th International Workshop on Neural-Symbolic Learning and Reasoning, July 3–5, La Certosa di Pontignano, Siena, Italy*, *3432*. https://ceur-ws.org/Vol-3432/paper28.pdf

Herron, D., Jiménez-Ruiz, E., & Weyde, T. (2025). On the Potential of Logic and Reasoning in Neurosymbolic Systems using OWL-based Knowledge Graphs. *Neurosymbolic Artificial Intelligence*.

Hitzler, P. (2021). Knowledge Graphs in Neuro-Symbolic AI [Presentation slides]. https://people.cs.ksu.edu/~hitzler/pub2/2021-10-AKBC.pdf

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., & Rudolph, S. (2012). *OWL 2 Web Ontology Language Primer* (W3C Recommendation). World Wide Web Consortium. http://www.w3.org/TR/owl2-primer/

Hitzler, P., Krötzsch, M., & Rudolph, S. (2010). *Foundations of Semantic Web Technologies* (3rd ed.). CRC Press.

Hitzler, P., & Sarker, M. K. (Eds.). (2021). *Neuro-symbolic artificial intelligence: The state of the art*. IOS Press. https://ebooks.iospress.nl/volume/neuro-symbolic-artificial-intelligence-the-state-of-the-art

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutiérrez, C., Kirrane, S., Labra Gayo, J. E., Navigli, R., Neumaier, S., Ngonga Ngomo, A.-C., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J. F., Staab, S., & Zimmermann, A. (2021). *Knowledge Graphs*. Springer. https://kgbook.org/

Horrocks, I., Kutz, O., & Sattler, U. (2006). The Even More Irresistible SROIQ. In P. Doherty, J. Mylopoulos, & C. A. Welty (Eds.), *Proceedings, tenth international conference on principles of knowledge representation and reasoning* (pp. 57–67). AAAI Press. http://www.aaai.org/Library/KR/2006/kr06-009.php

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member Submission, W3C*. https://www.w3.org/submissions/SWRL/

Huth, M., & Ryan, M. (2004). *Logic in Computer Science* (2nd ed.). Cambridge University Press.

Jackson, R., Matentzoglu, N., Overton, J. A., Vita, R., Balhoff, J. P., Buttigieg, P. L., Carbon, S., Courtot, M., Diehl, A. D., Dooley, D. M., Duncan, W. D., Harris, N. L., Haendel, M. A., Lewis, S. E., Natale, D. A., Osumi-Sutherland, D., Ruttenberg, A., Schriml, L. M., Smith, B., ... Peters, B. (2021). OBO Foundry in 2021: Operationalizing Open Data Principles to Evaluate Ontologies [http://obofoundry.org/]. *Database*, *2021*. https://doi.org/10.1093/database/baab069

John, T., & Koopmann, P. (2024). Planning with OWL-DL Ontologies. *ECAI 2024 - 27th European Conference on Artificial Intelligence*, *392*, 4165–4172. https://doi.org/10.3233/FAIA240988

Kahl, W., & Schmidt, G. (2000). *Exploring (Finite) Relation Algebras Using Tools Written in Haskell* (tech. rep.) (The RATH system). Federal Armed Forces University Munich. https://titurel.org/Papers/RelAlgTools.pdf

Kazakov, Y., Krötzsch, M., & Simančík, F. (2011). Unchain My *EL* Reasoner. *Proceedings of the 24th International Workshop on Description Logics (DL'11)*, *745*.

Keet, C. M. (2020). *An Introduction to Ontology Engineering* ('v1.5'). https://people.cs.uct.ac.za/~mkeet/OEbook/

Kejriwal, M., Knoblock, C. A., & Szekely, P. (2021). *Knowledge Graphs: Fundamentals, Techniques, and Applications*. The MIT Press.

Kendall, E. F., & McGuinness, D. L. (2019). *Ontology Engineering*. Morgan & Claypool Publishers.

Khan, M. J., Breslin, J. G., & Curry, E. (2022). Common Sense Knowledge Infusion for Visual Understanding and Reasoning: Approaches, Challenges, and Applications. *IEEE Internet Comput.*, *26*(4), 21–27. https://doi.org/10.1109/MIC.2022.3176500

Kipf, T. N., & Welling, M. (2016). Semi-Supervised Classification with Graph Convolutional Networks. *CoRR*, *abs/1609.02907*. http://arxiv.org/abs/1609.02907

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L., Shamma, D. A., Bernstein, M. S., & Fei-Fei, L. (2016). Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *CoRR*. http://arxiv.org/abs/1602.07332

Krötzsch, M., Simancik, F., & Horrocks, I. (2013). A Description Logic Primer. *CoRR*, *abs/1201.4089*. http://arxiv.org/abs/1201.4089

Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J. R. R., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., Duerig, T., & Ferrari, V. (2020). The Open Images Dataset V4. *Int. J. Comput. Vis.*, *128*(7), 1956–1981. https://doi.org/10.1007/s11263-020-01316-z

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., & Bizer, C. (2015). DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web*, *6*(2), 167–195. http://dx.doi.org/10.3233/SW-140134

Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. S. (2016). Gated Graph Sequence Neural Networks. *4th International Conference on Learning Representations, ICLR 2016*. http://arxiv.org/abs/1511.05493

Lin, T., Maire, M., Belongie, S. J., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. *Computer Vision - ECCV 2014*, *8693*, 740–755. https://doi.org/10.1007/978-3-319-10602-1%5C_48

Lu, C., Krishna, R., Bernstein, M., & Fei-Fei, L. (2016). Visual Relationship Detection with Language Priors. *European Conference on Computer Vision*, 852–869. https://cs.stanford.edu/people/ranjaykrishna/vrd/

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *1st International Conference on Learning Representations, ICLR*. http://arxiv.org/abs/1301.3781

Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., & Lutz, C. (2012). *OWL 2 Web Ontology Language Profiles* (W3C Recommendation). World Wide Web Consortium. https://www.w3.org/TR/owl2-profiles/

Motik, B., Patel-Schneider, P. F., Parsia, B., Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., & Smith, M. (2012). *OWL 2 web ontology language: Structural specification and functional-style*

syntax (W3C Recommendation). World Wide Web Consortium. http://www.w3.org/2007/OWL/draft/owl2-syntax/

Mouakher, A., Belkaroui, R., Bertaux, A., Labbani, O., Hugol-Gential, C., & Nicolle, C. (2019). An Ontology-Based Monitoring System in Vineyards of the Burgundy Region. *28th IEEE International Conference on Enabling Technologies*, 307–312. https://doi.org/10.1109/WETICE.2019.00070

Musen, M. A. (2015). The Protégé Project: A Look Back and a Look Forward. *AI Matters*, *1*(4), 4–12. https://doi.org/10.1145/2757001.2757003

Myklebust, E. B., Jiménez-Ruiz, E., Chen, J., Wolf, R., & Tollefsen, K. E. (2022). Prediction of adverse biological effects of chemicals using knowledge graph embeddings. *Semantic Web*, *13*(3), 299–338. https://doi.org/10.3233/SW-222804

Nakawala, H., Bianchi, R., Pescatori, L. E., Cobelli, O. D., Ferrigno, G., & Momi, E. D. (2019). "Deep-Onto" network for surgical workflow and context recognition. *International Journal of Computer Assisted Radiology Surgery*, *14*(4), 685–696. https://doi.org/10.1007/s11548-018-1882-8

Nardi, D., & Brachman, R. J. (2003). An Introduction to Description Logics. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, & P. F. Patel-Schneider (Eds.), *The description logic handbook: Theory, implementation, and applications* (pp. 1–40). Cambridge Univ. Press.

Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., & Banerjee, J. (2015). RDFox: A Highly-Scalable RDF Store [https://www.oxfordsemantic.tech/product]. *The Semantic Web - 14th International Conference, ISWC 2015, Proceedings*, *9367*, 3–20. https://ora.ox.ac.uk/objects/uuid:2a08b023-77be-431a-a08c-89b47381586a

Nickel, M., Murphy, K., Tresp, V., & Gabrilovich, E. (2015). A Review of Relational Machine Learning for Knowledge Graphs. *CoRR*. http://arxiv.org/abs/1503.00759

Nickel, M., Tresp, V., & Kriegel, H. (2011). A Three-Way Model for Collective Learning on Multi-Relational Data. In L. Getoor & T. Scheffer (Eds.), *Proceedings of the 28th international conference on machine learning, ICML* (pp. 809–816). Omnipress. https://icml.cc/2011/papers/438%5C_icmlpaper.pdf

Noy, N., & McGuinness, D. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology* (tech. rep.). Knowledge Systems Laboratory, Stanford University. http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf

Ontotext GraphDB [A popular RDF store; https://www.ontotext.com/products/graphdb/]. (2023). https://en.wikipedia.org/wiki/Ontotext_GraphDB

Ren, H., Hu, W., & Leskovec, J. (2020). Query2box: Reasoning over Knowledge Graphs in Vector Space Using Box Embeddings. *8th International Conference on Learning Representations, ICLR*. https://openreview.net/forum?id=BJgr4kSFDS

Ronchi, M. R., & Perona, P. (2015). Describing Common Human Visual Actions in Images [(COCO-a)]. In X. Xie, M. W. Jones, & G. K. L. Tam (Eds.), *Proceedings of the british machine vision conference (bmvc)* (pp. 52.1–52.12). BMVA Press. http://www.vision.caltech.edu/~mronchi/projects/Cocoa/

Rossi, A., Barbosa, D., Firmani, D., Matinata, A., & Merialdo, P. (2021). Knowledge Graph Embedding for Link Prediction: A Comparative Analysis. *ACM Trans. Knowl. Discov. Data*, *15*(2), 14:1–14:49. https://doi.org/10.1145/3424672

Russell, S., & Norvig, P. (2022). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson. https://aima.cs.berkeley.edu/global-index.html

Sarker, M. K., Zhou, L., Eberhart, A., & Hitzler, P. (2021). Neuro-Symbolic Artificial Intelligence: Current Trends. *CoRR*. https://arxiv.org/abs/2105.05330

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The Graph Neural Network Model. *IEEE Trans. Neural Networks*, *20*(1), 61–80. https://doi.org/10.1109/TNN.2008.2005605

Schlichtkrull, M. S., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., & Welling, M. (2018). Modeling Relational Data with Graph Convolutional Networks. *The Semantic Web - 15th International Conference, ESWC 2018*, *10843*, 593–607. https://doi.org/10.1007/978-3-319-93417-4%5C_38

Schmidt, G. (2003). *Relational Language* (tech. rep. No. 2003-05) (Version 2, 101 pages — aka Titurel). Federal Armed Forces University Munich. https://titurel.org/Papers/RelLangTex20041025.pdf

Schmidt, G. (2004). A Proposal for a Multilevel Relational Reference Language. *Journal on Relational Methods in Computer Science — JoRMiCS*, *1*, 314–338. https://www.cosc.brocku.ca/Faculty/Winter/JoRMiCS/Vol1/PDF/v1n13.pdf

Schmidt, G. (2011). *Relational Mathematics* (Vol. 132). Cambridge University Press. http://www.cambridge.org/de/knowledge/isbn/item2713741/

Schmidt, G., & Ströhlein, T. (1993). *Relations and Graphs - Discrete Mathematics for Computer Scientists*. Springer-Verlag.

Schmidt, G., & Winter, M. (2014). Relational Mathematics Continued. *CoRR, abs/1403.6957*. http://arxiv.org/abs/1403.6957

Sequeda, J., Allemang, D., & Jacob, B. (2025). Knowledge graphs as a source of trust for llm-powered enterprise question answering. *Journal of Web Semantics*, *85*. https://www.sciencedirect.com/science/article/pii/S1570826824000441

Serafini, L., & d'Avila Garcez, A. S. (2016). Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge. *Proceedings of the 11th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy'16), 1768*. http://ceur-ws.org/Vol-1768/NESY16%5C_paper3.pdf

Sheth, A. P., Gaur, M., Kursuncu, U., & Wickramarachchi, R. (2019). Shades of Knowledge-Infused Learning for Enhancing Deep Learning. *IEEE Internet Comput.*, *23*(6), 54–63. https://doi.org/10.1109/MIC.2019.2960071

Singhal, A. (2012). Introducing the Knowledge Graph: things, not strings [Google blog]. https://www.blog.google/%20products/search/introducing-knowledge-graph-things-not/

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A Practical OWL-DL Reasoner. *J. Web Semant.*, *5*(2), 51–53. https://doi.org/10.1016/j.websem.2007.03.004

Staab, S., & Studer, R. (Eds.). (2009). *Handbook on Ontologies* (2nd ed.). Springer.

Sun, Z., Deng, Z., Nie, J., & Tang, J. (2019). RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. *7th International Conference on Learning Representations, ICLR*. https://openreview.net/forum?id=HkgEQnRqYQ

Tanon, T. P., Weikum, G., & Suchanek, F. M. (2020). YAGO 4: A Reasonable Knowledge Base. *The Semantic Web - 17th International Conference, ESWC 2020, Proceedings*, *12123*, 583–596. https://doi.org/10.1007/978-3-030-49461-2%5C_34

Tarski, A. (1941). On the Calculus of Relations. *Journal of Symbolic Logic*, *6*(3), 73–89. https://doi.org/10.2307/2268577

Trinh, T. H., Wu, Y., Le, Q. V., He, H., & Luong, T. (2024). Solving olympiad geometry without human demonstrations. *Nature*, *625*, 476–482. https://doi.org/10.1038/s41586-023-06747-5

Uschold, M. (2018). *Demystifying OWL for the Enterprise*. Morgan; Claypool.

van Bekkum, M., de Boer, M., van Harmelen, F., Meyer-Vitali, A., & ten Teije, A. (2021). Modular Design Patterns for Hybrid Learning and Reasoning Systems. *Appl. Intell.*, *51*(9).

van Harmelen, F., & ten Teije, A. (2019). A Boxology of Design Patterns for Hybrid Learning and Reasoning Systems. *J. Web Eng.*, *18*(1-3), 97–124.

Vashishth, S., Sanyal, S., Nitin, V., & Talukdar, P. P. (2020). Composition-based Multi-Relational Graph Convolutional Networks. *8th International Conference on Learning Representations, ICLR 2020*. https://openreview.net/forum?id=BylA%5C_C4tPr

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2017). Graph Attention Networks. *CoRR*, *abs/1710.10903*. http://arxiv.org/abs/1710.10903

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

Wang, M., Qiu, L., & Wang, X. (2021). A Survey on Knowledge Graph Embeddings for Link Prediction. *Symmetry*, *13*(3). https://doi.org/10.3390/sym13030485

Warshall, S. (1962). A Theorem on Boolean Matrices. *J. ACM*, *9*(1), 11–12. https://doi.org/10.1145/321105.321107

Whetzel, P. L., Noy, N. F., Shah, N. H., Alexander, P. R., Nyulas, C., Tudorache, T., & Musen, M. A. (2011). BioPortal: Enhanced Functionality via new Web Services from the National Center for Biomedical Ontology to Access and use Ontologies in Software Applications [https://bioportal.bioontology.org/]. *Nucleic Acids Research*, *39*(suppl 2), W541–W545.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Networks Learn. Syst.*, *32*(1), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

Zhang, Z., Cui, P., & Zhu, W. (2022). Deep Learning on Graphs: A Survey. *IEEE Trans. Knowl. Data Eng.*, *34*(1), 249–270. https://doi.org/10.1109/TKDE.2020.2981333

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph Neural Networks: A Review of Methods and Applications. *AI Open*, *1*, 57–81. https://doi.org/10.1016/j.aiopen.2021.01.001

Zhu, G., Zhang, L., Jiang, Y., Dang, Y., Hou, H., Shen, P., Feng, M., Zhao, X., Miao, Q., Shah, S. A. A., & Bennamoun, M. (2022). Scene Graph Generation: A Comprehensive Survey. *CoRR*. https://arxiv.org/abs/2201.00443

Appendix A

Our VRD-World Wikidata-inspired class hierarchy

Figure A.1 shows our Wikidata-inspired class hierarchy for the NeSy4VRD domain. At the time of creation (2022), this was a faithful, tiny subset of the Wikidata class hierarchy, but for one minor exception: Wikidata entity 'giraffe' (Q862089) was an *instance*, but we pretended it was a *class* and modelled it as a subclass of Wikidata entity 'mammal' (Q7377). It was a candidate class hierarchy for our VRD-World ontology, but it has yet to be formalised in that way.

We could not find a description of the Wikidata class hierarchy, so we devised a process for discovering it for ourselves—or, at least, the portions of it relevant to our needs. Our objective was to have NeSy4VRD object classes as leaf classes, with a unifying hierarchy of Wikidata parent classes above them.

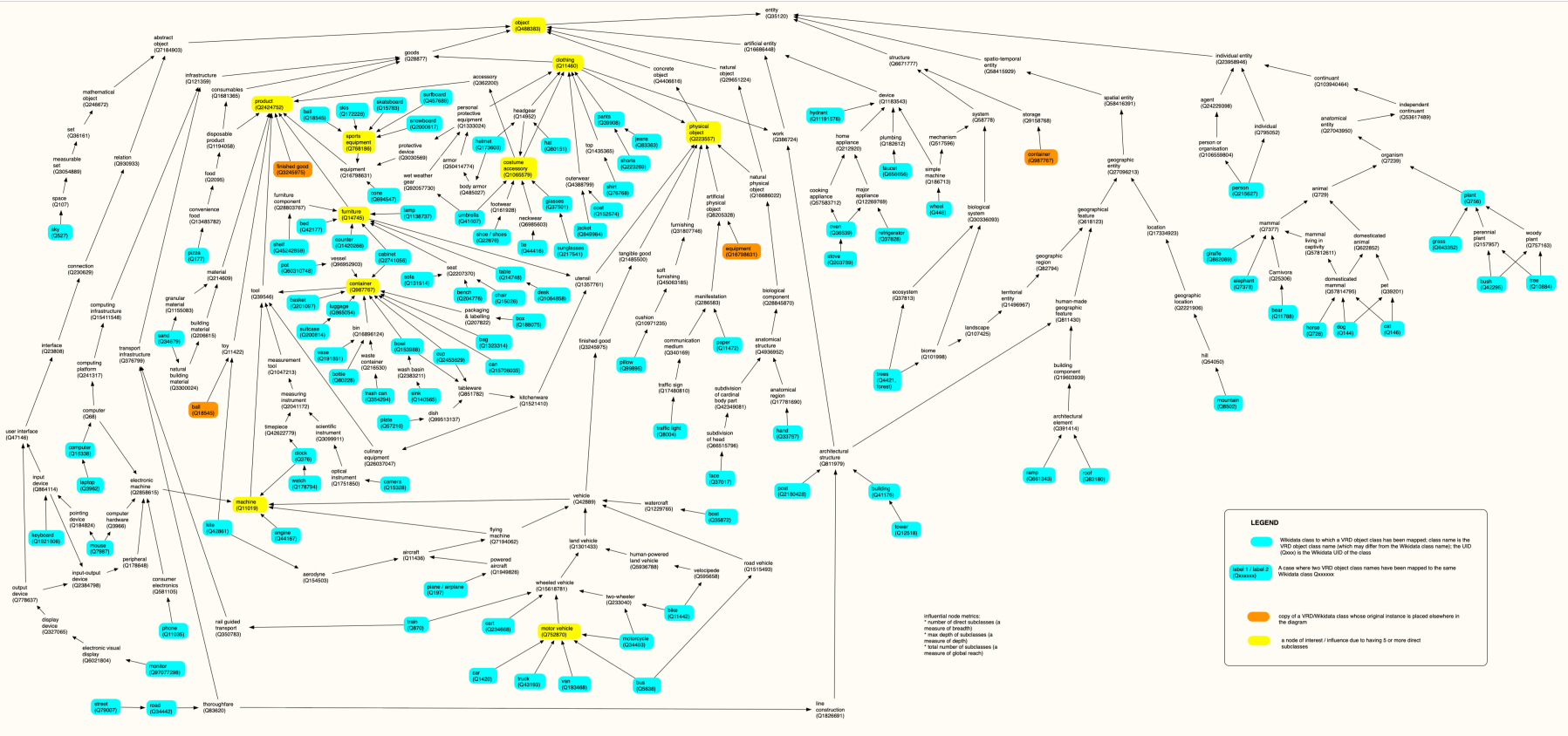


Figure A.1: A pictorial rendering of a potential alternate class hierarchy for the VRD-world OWL ontology, inspired by the publicly available Wikidata knowledge graph. Classes coloured blue represent Wikidata classes (entities, with their Wikidata 'Qnnnnn' numbers) to which a NeSy4VRD object class name has been assigned. All other classes have their Wikidata names. Classes coloured orange are duplicates introduced to avoid clutter from criss-crossing arrows. All arrows represent Wikidata `subClassOf` relationships. The colour yellow highlights classes with many subclasses and/or parents.

To discover the Wikidata class hierarchy, and to extract hierarchy paths of relevance to our NeSy4VRD object classes, we devised a semi-automated process. The steps of this process, for each NeSy4VRD object class, were as follows:

1. We mapped the NeSy4VRD object class to a Wikidata entity. To do this, we first used SPARQL queries to find candidate matching entities based on word matches between the NeSy4VRD object class name and *(i)* Wikidata entity names, and *(ii)* the literal values of Wikidata entity 'label', 'description' and 'also known as' attributes. Then, using manual inspection of the candidate matches, we selected a 'best matching' Wikidata entity (class), `X`.

2. We ran a SPARQL query that we developed for Wikidata which, given entity `X`, returns every `[A,B]` child/parent relation (link) involved in every subsumption chain connecting `X` to the top-level Wikidata entity. The result set was a 'bag' of `[A,B]` child/parent relations.

3. We ran a recursive algorithm we developed that takes the bag of atomic `[A,B]` subsumption chain relations for entity `X` as input, and returns every unique, complete subsumption chain leading from `X` to the top-level Wikidata entity. (Interestingly, the numbers of unique, complete chains for different entities `X` ranged from 1, for class 'elephant', to 169, for class 'laptop'.)

4. We then browsed the unique subsumption chains for entity `X` and selected a small number (1, 2 or 3) that looked interesting, credible, and which fit well with our gradually emerging class hierarchy. We then extended our class hierarchy by 'patching in' entity `X` and the (usually few) additional Wikidata entities (classes) required to completely represent the selected subsumption chains for `X`.

We iterated over this process for each NeSy4VRD object class. At the end of the process we had a subset of the Wikidata class hierarchy that incorporated each of the NeSy4VRD object classes.

Critique of the Wikidata class hierarchy In the process of extracting our subset of the Wikidata class hierarchy for potential use as a class hierarchy in our VRD-World OWL ontology, we learned much about the Wikidata class hierarchy itself. It is both impressive and disappointing. Our overall impression was of a heaving, chaotic mess; a crowd-sourced free-for-all. Sometimes it was ludicrous. We had the sense that there was no governing vision or mechanism for managing the quality of the hierarchy. The message is: ontology design and crowd-sourcing do not appear to mix well.

To illustrate our point, we offer the reader a few entertaining portions (sub-chains) of Wikidata subsumption chains, expressed as sequences of `[A,B]` links, where `A` is a subclass of `B`.

1) The following sub-chain of the Wikidata class hierarchy asserts that a ‘stove’ is a ‘geographical feature’:

```
['Q203789', 'stove', 'Q36539', 'oven']
['Q36539', 'oven', 'Q391414', 'architectural element']
['Q391414', 'architectural element', 'Q618123', 'geographical feature']
```

2) This sub-chain asserts that a ‘sink’ is ‘tableware’ and a ‘utensil’:

```
['Q140565', 'sink', 'Q2383211', 'wash basin']
['Q2383211', 'wash basin', 'Q153988', 'bowl']
['Q153988', 'bowl', 'Q851782', 'tableware']
['Q851782', 'tableware', 'Q1521410', 'kitchenware']
['Q1521410', 'kitchenware', 'Q1357761', 'utensil']
```

3) Finally, this sub-chain asserts that a computer ‘keyboard’ is a ‘person’:

```
['Q1921606', 'keyboard', 'Q864114', 'input device']
['Q864114', 'input device', 'Q1339255', 'receiver']
['Q1339255', 'receiver', 'Q4425763', 'interlocutor']
['Q4425763', 'interlocutor', 'Q215627', 'person']
```

We avoided modelling sub-chains (1) and (3) in our candidate Wikidata-inspired class hierarchy for VRD-World, but sub-chain (2) is there, in Figure A.1, if one looks hard enough.

Appendix B

Platform Benchmarking Results

The pages which follow this one contain a copy of a short report we wrote describing a platform benchmarking exercise we conducted in the course of our research. The objective of the exercise was to choose a platform for conducting many of our experiments. The findings give mean per epoch training times for a specific neural network training task across a variety of platforms: personal consumer-grade machines, the Hyperion HPC cluster at City St George's, University of London, and several Amazon Web Services (AWS) EC2 instance types.

Platform Benchmarking Exercise for PhD Research with Attention to City's Hyperion

Dave Herron

February 2024

Introduction

This document reports findings from a platform benchmarking exercise undertaken in December, 2023, in relation to PhD research. The objective was to gather data with which to model scenarios that deliver different levels of experiment throughput per day, at different associated costs, for research planning purposes.

Towards this end, a single neural network training script (job) was used to measure per epoch training times on several different platforms. The training job trains a small, 5-layer, feed-forward PyTorch classification neural network. The training data is flat, feature vectors of size 239. The training mini-batches are small and they vary in size; some have a few feature vectors, others many dozens. (This detail may be relevant to understanding some of the results.) All environments used Python 3.11.5 and PyTorch 2.0.1.

Results

Table 1 reports mean per epoch training times by platform when just a single instance of the training job is running on a given platform. All times are in minutes. The platforms included in this stage of the benchmarking exercise fall into three categories: 1) personal machines (owned by the author), 2) City's Hyperion HPC cluster, and 3) Amazon Web Services EC2 instance types of various size and shape.

Table 2 reports mean per epoch training times by platform when different numbers of instances of the training job are running in parallel on a given platform. Again, all times are in minutes. The platforms included in this stage of the benchmarking exercise fall into two categories: 1) personal machines (owned by the author), 2) Amazon Web Services EC2 instance types of various size and shape.

Discussion

Here we highlight and discuss observations revealed by the results.

Discussion per Table 1

If we focus on the results in Table 1 that relate to the author's personal machines, we can see that the training times are slightly better when running the training job on the machine's CPU rather than on its GPU. I think this is a quirk of my neural network model and training data — and a sign that they are not large enough or complex enough to get benefit from GPUs. As explained, the mini-batches are small and vary in size, so perhaps the overheads of copying data between the CPU and GPU outweigh the meagre computational benefits once on the GPU. At any rate, the training job appears to be CPU-intensive rather than GPU-intensive. But this fact does not prevent us from observing how City's Hyperion platform compares with the other platforms in relative terms. We are just explaining an apparent anomaly in the results to help readers avoid becoming confused.

The results delivered by the author's personal machines compare favourably with those delivered by City's Hyperion. The two Apple Silicon machines are more than twice as fast. Even the author's Linux laptop, purchased in 2020, can match Hyperion. The CPU of that laptop is an Intel Core i7-9750H @ 2.6GHz; its GPU is an Nvidia GeForce GTX 1650 GPU with 4GB of memory (effectively obsolete by now).

For fun, we used one trial of our benchmarking exercise to run our training job on the CPU of the 'gengpu' node of Hyperion (something that would normally never be done). As expected, the per epoch training times increased dramatically (to 37.5 minutes per epoch). This is explained by the fact that this CPU would have been busy managing several other users' jobs running on (sharing) the 'gengpu' node (and its GPUs) when we ran our trial.

We ran trials on several different Amazon Web Services EC2 instance types. Surprisingly, and discouragingly, we did not find attractive GPU options on AWS. Their 'p3' instance types use Nvidia V100 GPUs (the predecessor of the A100). This generation of GPU cannot match the throughput of the A100 on Hyperion, or match the author's personal machines. The AWS 'p4' instance types offer access to a full A100 GPU, meaning one has access to up to all 8 GPUs on an A100 unit. But the AWS cost, at $35 US/hour, is prohibitive. We declined to run a trial given that our training job is not GPU-intensive, and given the performance indications from Hyperion.

We tested several EC2 instance types running either AMD CPUs (c6a, c7a) or Intel CPUs (c6i, c7i). Interestingly, the AMD CPUs deliver slightly faster training times than their Intel counterparts. But this is when just a single instance of the training job is running. The situation changes when running multiple instances in parallel (see Table 2).

Lastly, AWS provides access to Apple Mac machines comparable to those owned by the author. AWS categorises these as 'dedicated hosts' and rents them on a different basis to its 'shared' EC2 instance types. One learns the hard way that once allocated to your account, a dedicated host cannot be unallocated until at least 24 hours has passed. So the minimum charge for using a Mac Mini, with an M2 Pro chip, is $24 \times 1.56 = \$37.44$, which makes it prohibitively expensive. But the training time performance is identical to what the author's Apple machines deliver, which is reassuring.

Discussion per Table 2

In Table 2, first notice that Hyperion does not feature. This highlights the fact that, on a platform like Hyperion, we do not have the freedom to launch multiple instances of a job on the same node. So we cannot run the trials reported in Table 2 on Hyperion. When using the 'nodes' partition of Hyperion, each job one submits to SLURM gets allocated to a different node. On the 'gengpu' partition, one can submit multiple jobs to run the same script, and these will all run on the 'gengpu' node in parallel, but there is a (necessary) limit on the number of jobs per user, so this freedom to run jobs in parallel is tightly restricted. In contrast, in a cloud environment like AWS, there is no job submission and no job queuing. Once one's EC2 instance is running, you can open as many Terminal session connections to it as you want, and launch as many instances of your job as that platform can reasonably sustain. This suggests there may be a lot of horsepower on Hyperion that simply cannot be leveraged and is going to waste. Each node in the 'nodes' partition might be capable of running several instances of a job in

Table 1: **Mean per epoch training times (in minutes) by platform, with a single job instance running**

Platform	**OS**	**CPU**	**Apple GPU**	**Nvidia GPU**	**Cost**
PyTorch device		'cpu'	'mps'	'cuda'	($/hr)
Personal machines					
MacBook Pro 16" (M2 Pro)	Sonoma	6.0	6.9		
Mac Studio (M2 Ultra)	Sonoma	6.2	6.8		
Linux laptop (GeForce GPU)	Ubuntu	14.9		15.4	
City's Hyperion					
partition 'gengpu' (A100 GPU)	Ubuntu	37.5		15.1	
partition 'nodes' (CPU only)	Ubuntu	16.3			
Amazon Web Services					
p3.2xlarge (V100 GPU)	Ubuntu			26.2	3.59
p4d.24xlarge (full A100 GPU)	Ubuntu			n/a	35.00
c6a.2xlarge (AMD CPU)	Ubuntu	12.6			
c6i.2xlarge (Intel CPU)	Ubuntu	14.8			
c6i.4xlarge (Intel CPU)	Ubuntu	14.6			
c7a.xlarge (AMD CPU)	Ubuntu	8.2			
c7a.2xlarge (AMD CPU)	Ubuntu	8.1			0.41
c7a.4xlarge (AMD CPU)	Ubuntu	8.2			
c7i.2xlarge (Intel CPU)	Ubuntu	11.8			0.36
c7i.4xlarge (Intel CPU)	Ubuntu	9.9			0.72
mac2-m2pro.metal (M2 Pro)	Sonoma		6.9		1.56

parallel, but there's no way to make that happen.

Table 2 shows that, as expected, per epoch training times increase with the number of job instances running in parallel. But, on the whole, the degradation is measured and gradual, at least until the platform gets maxxed-out. The value of the Apple Silicon GPU shows itself here, when multiple jobs are running. The 'mps' backend (the GPUs and the neural engine) allow larger numbers of jobs to run in parallel comfortably.

Interestingly, the AWS EC2 instance types c7a.2xlarge and c7a.4xlarge, both running AMD CPUs, could not support a 2nd instance of our training job. When a single instance of our training job runs on these AMD CPUs, the per epoch training times are faster than the Intel counterparts. But if we run a 2nd instance, the machine gets absurdly maxxed-out and training times explode. In contrast, the c7i instance types, running Intel CPUs, can handle multiple instances comfortably. But only up to a point. The c7i.4xlarge instance type handled 5 and 6 instances of our job comfortably, but gets maxxed-out at 7 instances — far fewer than the Mac Studio with M2 Ultra chip can handle.

The author's platform preferences

Given the results in Tables 1 and 2, for the category of research work in question, the author is inclined to rely on his two personal Apple Silicon machines as much as possible.

Table 2: **Mean per epoch training times (in minutes) by platform, with multiple identical job instances running in parallel**

Jobs	personal MBP16 M2 Pro 'cpu'	personal MPB16 M2 Pro 'mps'	personal MStudio M2 Ultra 'cpu'	personal MStudio M2 Ultra 'mps'	AWS MacMini M2 Pro 'mps'	AWS c7a.2x CPU 'cpu'	AWS c7a.4x CPU 'cpu'	AWS c7i.4x CPU 'cpu'
1	6.0	6.9	6.2	6.8	6.9	8.1	8.2	9.9
2	6.2	7.1	7.4	7.2	7.0	38.1	39.4	9.8
3	6.4	7.2	8.6	7.2	7.2			10.1
4	6.5	7.3	11.9	7.3	7.2			10.1
5	7.1	7.4	12.4	7.4	7.3			10.3
6		7.5	13.5	7.5	7.4			11.1
7		7.8		7.5	7.4			17.4
8				7.6	7.7			
9				7.6	8.4			
10				7.7				
11				...				

If more experiment throughput-per-day is needed, the next most attractive option is a toss-up. The AWS EC2 instance type c7i.4xlarge supports 4 to 6 jobs in parallel comfortably, at reasonable speed, at a cost of $US 0.72/hour. The 'nodes' partition of City's Hyperion can reliably support 4 to 6 jobs in parallel at no cost, but with substantially longer training times.

Additional points

AWS has further categories of EC2 instance types and processor technologies that were not examined as part of this benchmarking exercise. For examples. Amazon makes its own Graviton chip. The price performance of Graviton may be attractive, but we have not looked at it.

The Apple Silicon GPUs and Neural Engine are opaque. The ability to monitor GPU usage on Mac is still limited. And the Neural Engine is a total black box. There is zero visibility as to whether, when and to what extent it is engaged, if ever. This is disappointing.

If we compare the user's 'computing experience' on the two cloud platforms, Hyperion and AWS, we prefer the user experience on AWS. One feels more in control on a platform like AWS, largely because there is no job management infrastructure (like SLURM) mediating your experience. You have direct access to your compute node and, as described earlier, it is easy and natural to open multiple sessions to it and run several jobs in parallel. Also, there's no queuing for scarce, shared resources. On Hyperion, one often needs to wait to get access to 'gengpu'. The 'nodes' partition is different — access there is easier.

Platform specification details

MacBook Pro 16", M2 Pro

- 3.5 GHz clock speed
- 12 core CPU (4 power-efficiency, 8 performance)
- 19 core GPU; 16 core Neural Engine

Mac Studio, M2 Ultra

- 3.7 GHz clock speed
- 24 core CPU
- 60 core GPU; 32 core Neural Engine

Linux laptop

- CPU: Intel Core i7-9750H CPU @ 2.6GHz; 6 cores, 2 threads per core
- GPU: Nvidia GeForce GTX 1650, 4GB memory

City's Hyperion

- each HPC compute node has 2 Intel Xeon Gold 6248R CPU @ 3.00GHz processors; each processor has 24 cores, with 2 threads per core, giving 48 threads
- in addition, the 'gengpu' node has an Nvidia A100 unit, which is 8 A100 GPUs

AWS c6a.2xlarge and c6a.4xlarge (AMD CPU)

- 3rd generation AMD EPYC processors (AMD EPYC 7R13); 3.6 GHz

AWS c6i.2xlarge and c6i.4xlarge (Intel CPU)

- 3rd generation Intel Xeon Scalable processors (Ice Lake 8375C); 3.5 GHz

AWS c7a.2xlarge and c7a.4xlarge (AMD CPU)

- 4th generation AMD EPYC processors (AMD EPYC 9R14); 3.7 GHz

AWS c7i.2xlarge and c7i.4xlarge (Intel CPU)

- 4th generation Intel Xeon Scalable processors (Sapphire Rapids 8488C); 3.2 GHz
- 2xlarge has 4 CPU cores with 2 threads per core = 8 vCPUs
- 4xlarge has 8 CPU cores with 2 threads per core = 16 vCPUs

Appendix C

Our mini VRD-World ontology

Listing C.1 shows mini VRD-World OWL ontology. It is a representative subset of the much larger VRD-World OWL ontology. By design, it packs a wide breadth of OWL inference semantics into a small data space, in order to facilitate efficient development of our tensor knowledge graph reasoning techniques.

Listing C.1: Our mini VRD-World ontology.

```
@prefix : <http://example.com/tkgre/mini_vrd_world#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@base <http://example.com/tkgre/mini_vrd_world#> .

<http://example.com/tkgre/mini_vrd_world> rdf:type owl:Ontology ;
dc:description """An ontology designed to drive research and
development of our concepts of tensor knowledge graphs and tensor
knowledge graph reasoning within our emerging tensor knowledge
graph reasoning engine.

This ontology is loosely based on the VRD-World v1 ontology. It
represents a small, adapted subset of VRD-World, with a much smaller,
simpler class hierarchy and many fewer object properties.  Some
entities do not appear in VRD-World and have been introduced for
convenience.

We take liberties with the names of things and the semantics of
things in order to pack as much inference semantics as we can into
a small ontological space. That is, we deliberately arrange for a
high concentration of OWL inference semantics. We arrange for the
breadth of inference semantics to be sufficient to cover those
aspects of OWL that we seek to represent, and those aspects of
OWL reasoning that we seek to emulate.

Unlike the VRD-World ontology, mini VRD-World includes individuals,
and data triples involving those individuals. These individuals and
data triples have been arranged (conceived) in order to efficiently
drive tensor knowledge graph research and development.  Individuals
are generally named by the class they belong to followed by an
arbitrary 3-digit sequence number, to facilitate interpretation
of what the tensor knowledge graph reasoning is doing.

The ontology is fluid. We adapt and extend it as required based on
current research and development priorities.""" ;
dc:title "Tensor KG Reasoning Engine research and development ontology" ;
```

```
dcterms:contributor "David Herron" ;
dcterms:license "Creative Commons Attribution 4.0 (CC BY 4.0)" ;
rdfs:label "Mini VRD-World"@en ;
owl:versionInfo "1.0" .

#################################################################
#    Annotation properties
#################################################################

###  http://purl.org/dc/elements/1.1/description
dc:description rdf:type owl:AnnotationProperty .


###  http://purl.org/dc/elements/1.1/title
dc:title rdf:type owl:AnnotationProperty .


###  http://purl.org/dc/terms/contributor
dcterms:contributor rdf:type owl:AnnotationProperty .


###  http://purl.org/dc/terms/license
dcterms:license rdf:type owl:AnnotationProperty .


#################################################################
#    Object Properties
#################################################################

###  http://example.com/tkgre/mini_vrd_world#above
:above rdf:type owl:ObjectProperty ;
       rdfs:subPropertyOf :higherUpThan ;
       owl:inverseOf :below ;
       rdf:type owl:AsymmetricProperty ,
                owl:TransitiveProperty .


###  http://example.com/tkgre/mini_vrd_world#below
:below rdf:type owl:ObjectProperty ;
       rdfs:subPropertyOf :lowerDownThan ;
       rdf:type owl:AsymmetricProperty ,
                owl:TransitiveProperty .


###  http://example.com/tkgre/mini_vrd_world#beneath
:beneath rdf:type owl:ObjectProperty ;
         owl:equivalentProperty :under ;
         owl:inverseOf :over ;
         rdf:type owl:TransitiveProperty .


###  http://example.com/tkgre/mini_vrd_world#beside
:beside rdf:type owl:ObjectProperty ;
        owl:equivalentProperty :nextTo .


###  http://example.com/tkgre/mini_vrd_world#eat
:eat rdf:type owl:ObjectProperty ;
     rdfs:subPropertyOf :has ;
     rdfs:domain :Mammal ;
     rdfs:range :Food .


###  http://example.com/tkgre/mini_vrd_world#has
:has rdf:type owl:ObjectProperty .


###  http://example.com/tkgre/mini_vrd_world#hasBbox
:hasBbox rdf:type owl:ObjectProperty ,
                  owl:FunctionalProperty ,
                  owl:InverseFunctionalProperty ;
         rdfs:domain :VRDworldThing ;
         rdfs:range :Bbox .


###  http://example.com/tkgre/mini_vrd_world#higherUpThan
:higherUpThan rdf:type owl:ObjectProperty ;
              rdfs:subPropertyOf :higherUpThan2 .


###  http://example.com/tkgre/mini_vrd_world#higherUpThan2
:higherUpThan2 rdf:type owl:ObjectProperty .
```

```
###  http://example.com/tkgre/mini_vrd_world#hold
:hold rdf:type owl:ObjectProperty ;
      rdfs:subPropertyOf :has ;
      rdfs:domain :HoldCapableThing .


###  http://example.com/tkgre/mini_vrd_world#irreflexiveProp
:irreflexiveProp rdf:type owl:ObjectProperty ,
                          owl:IrreflexiveProperty .


###  http://example.com/tkgre/mini_vrd_world#lowerDownThan
:lowerDownThan rdf:type owl:ObjectProperty .


###  http://example.com/tkgre/mini_vrd_world#near
:near rdf:type owl:ObjectProperty ,
               owl:SymmetricProperty .


###  http://example.com/tkgre/mini_vrd_world#nextTo
:nextTo rdf:type owl:ObjectProperty ,
                 owl:SymmetricProperty .


###  http://example.com/tkgre/mini_vrd_world#on
:on rdf:type owl:ObjectProperty .


###  http://example.com/tkgre/mini_vrd_world#over
:over rdf:type owl:ObjectProperty ;
      rdfs:subPropertyOf :above ;
      owl:inverseOf :under ;
      rdf:type owl:TransitiveProperty .


###  http://example.com/tkgre/mini_vrd_world#reflexiveProp
:reflexiveProp rdf:type owl:ObjectProperty ,
                        owl:ReflexiveProperty .


###  http://example.com/tkgre/mini_vrd_world#ride
:ride rdf:type owl:ObjectProperty ;
      rdfs:subPropertyOf :on ;
      rdfs:domain :Person ;
      rdfs:range :Bike .


###  http://example.com/tkgre/mini_vrd_world#under
:under rdf:type owl:ObjectProperty ;
       rdfs:subPropertyOf :below ;
       rdf:type owl:TransitiveProperty .


###  http://example.com/tkgre/mini_vrd_world#wear
:wear rdf:type owl:ObjectProperty ;
      rdfs:subPropertyOf :has ;
      rdfs:domain :WearCapableThing ;
      rdfs:range :WearableThing .


#################################################################
#    Data properties
#################################################################

###  http://example.com/tkgre/mini_vrd_world#hasCoordinateXmax
:hasCoordinateXmax rdf:type owl:DatatypeProperty ,
                            owl:FunctionalProperty ;
                   rdfs:domain :Bbox ;
                   rdfs:range xsd:integer .


###  http://example.com/tkgre/mini_vrd_world#hasCoordinateXmin
:hasCoordinateXmin rdf:type owl:DatatypeProperty ,
                            owl:FunctionalProperty ;
                   rdfs:domain :Bbox ;
                   rdfs:range xsd:integer .


###  http://example.com/tkgre/mini_vrd_world#hasCoordinateYmax
:hasCoordinateYmax rdf:type owl:DatatypeProperty ,
                            owl:FunctionalProperty ;
```

```
                  rdfs:domain :Bbox ;
                  rdfs:range xsd:integer .


###  http://example.com/tkgre/mini_vrd_world#hasCoordinateYmin
:hasCoordinateYmin rdf:type owl:DatatypeProperty ,
                           owl:FunctionalProperty ;
                  rdfs:domain :Bbox ;
                  rdfs:range xsd:integer .


#################################################################
#    Classes
#################################################################

###  http://example.com/tkgre/mini_vrd_world#Animal
:Animal rdf:type owl:Class ;
        rdfs:subClassOf :NaturalLivingThing .


###  http://example.com/tkgre/mini_vrd_world#Ball
:Ball rdf:type owl:Class ;
      rdfs:subClassOf :Toy ;
      owl:disjointWith :WearCapableThing .


###  http://example.com/tkgre/mini_vrd_world#Bbox
:Bbox rdf:type owl:Class ;
      rdfs:subClassOf :VRDworldInfrastructure .


###  http://example.com/tkgre/mini_vrd_world#Bicycle
:Bicycle rdf:type owl:Class ;
         owl:equivalentClass :Bike ;
         rdfs:subClassOf :NonMotorisedVehicle ;
         owl:disjointWith :NonBike .


###  http://example.com/tkgre/mini_vrd_world#Bike
:Bike rdf:type owl:Class ;
      rdfs:subClassOf :NonMotorisedVehicle ;
      owl:disjointWith :NonBike .


###  http://example.com/tkgre/mini_vrd_world#Brolly
:Brolly rdf:type owl:Class ;
        owl:equivalentClass :Umbrella ;
        rdfs:subClassOf :Device .


###  http://example.com/tkgre/mini_vrd_world#Car
:Car rdf:type owl:Class ;
     rdfs:subClassOf :MotorisedVehicle .


###  http://example.com/tkgre/mini_vrd_world#Carnivore
:Carnivore rdf:type owl:Class ;
           rdfs:subClassOf :Mammal ;
           owl:disjointWith :Person .


###  http://example.com/tkgre/mini_vrd_world#Cat
:Cat rdf:type owl:Class ;
     rdfs:subClassOf :Carnivore ;
     owl:disjointWith :HoldCapableThing ,
                      :WearCapableThing .


###  http://example.com/tkgre/mini_vrd_world#Clothing
:Clothing rdf:type owl:Class ;
          rdfs:subClassOf :WearableThing .


###  http://example.com/tkgre/mini_vrd_world#Device
:Device rdf:type owl:Class ;
        rdfs:subClassOf :EngineeredThing ,
                        :NonBike ,
                        :NonWearableThing ;
        owl:disjointWith :WearCapableThing .


###  http://example.com/tkgre/mini_vrd_world#Dog
:Dog rdf:type owl:Class ;
```

```
     rdfs:subClassOf :Carnivore .


###  http://example.com/tkgre/mini_vrd_world#EngineeredThing
:EngineeredThing rdf:type owl:Class ;
                 rdfs:subClassOf :NonFood ,
                                 :VRDworldThing ;
                 owl:disjointWith :HoldCapableThing ,
                                  :Mammal ,
                                  :Person .


###  http://example.com/tkgre/mini_vrd_world#Food
:Food rdf:type owl:Class ;
      rdfs:subClassOf :MixedThing ,
                      :NonBike ,
                      :NonWearableThing ;
      owl:disjointWith :HoldCapableThing ,
                       :NonFood ,
                       :WearCapableThing .


###  http://example.com/tkgre/mini_vrd_world#Hat
:Hat rdf:type owl:Class ;
     rdfs:subClassOf :UpperBodyClothing .


###  http://example.com/tkgre/mini_vrd_world#HoldCapableThing
:HoldCapableThing rdf:type owl:Class ;
                  owl:equivalentClass [ rdf:type owl:Class ;
                                        owl:unionOf ( :Dog
                                                      :Person
                                                    )
                                      ] ;
                  rdfs:subClassOf :NaturalThing ;
                  owl:disjointWith :NaturalNonLivingThing .


###  http://example.com/tkgre/mini_vrd_world#Jeans
:Jeans rdf:type owl:Class ;
       rdfs:subClassOf :LowerBodyClothing .


###  http://example.com/tkgre/mini_vrd_world#LowerBodyClothing
:LowerBodyClothing rdf:type owl:Class ;
                   rdfs:subClassOf :Clothing .


###  http://example.com/tkgre/mini_vrd_world#Mammal
:Mammal rdf:type owl:Class ;
        rdfs:subClassOf :Animal ;
        owl:disjointWith :NaturalNonLivingThing .


###  http://example.com/tkgre/mini_vrd_world#MixedThing
:MixedThing rdf:type owl:Class ;
            rdfs:subClassOf :VRDworldThing .


###  http://example.com/tkgre/mini_vrd_world#MotorisedVehicle
:MotorisedVehicle rdf:type owl:Class ;
                  rdfs:subClassOf :NonBike ,
                                  :Vehicle .


###  http://example.com/tkgre/mini_vrd_world#NaturalLivingThing
:NaturalLivingThing rdf:type owl:Class ;
                    rdfs:subClassOf :NaturalThing .


###  http://example.com/tkgre/mini_vrd_world#NaturalNonLivingThing
:NaturalNonLivingThing rdf:type owl:Class ;
                       rdfs:subClassOf :NaturalThing ;
                       owl:disjointWith :Person ,
                                        :WearCapableThing .


###  http://example.com/tkgre/mini_vrd_world#NaturalThing
:NaturalThing rdf:type owl:Class ;
              rdfs:subClassOf :NonBike ,
                              :NonFood ,
                              :NonWearableThing ,
                              :VRDworldThing .
```

```
###  http://example.com/tkgre/mini_vrd_world#NonBike
:NonBike rdf:type owl:Class ;
         rdfs:subClassOf :MixedThing .


###  http://example.com/tkgre/mini_vrd_world#NonFood
:NonFood rdf:type owl:Class ;
         rdfs:subClassOf :MixedThing .


###  http://example.com/tkgre/mini_vrd_world#NonMotorisedVehicle
:NonMotorisedVehicle rdf:type owl:Class ;
                     rdfs:subClassOf :Vehicle .


###  http://example.com/tkgre/mini_vrd_world#NonWearableThing
:NonWearableThing rdf:type owl:Class ;
                  rdfs:subClassOf :MixedThing ;
                  owl:disjointWith :WearableThing .


###  http://example.com/tkgre/mini_vrd_world#Person
:Person rdf:type owl:Class ;
        rdfs:subClassOf :Primate .


###  http://example.com/tkgre/mini_vrd_world#Pizza
:Pizza rdf:type owl:Class ;
       rdfs:subClassOf :Food .


###  http://example.com/tkgre/mini_vrd_world#Primate
:Primate rdf:type owl:Class ;
         rdfs:subClassOf :Mammal .


###  http://example.com/tkgre/mini_vrd_world#Shirt
:Shirt rdf:type owl:Class ;
       rdfs:subClassOf :UpperBodyClothing .


###  http://example.com/tkgre/mini_vrd_world#Shoes
:Shoes rdf:type owl:Class ;
       rdfs:subClassOf :LowerBodyClothing .


###  http://example.com/tkgre/mini_vrd_world#Sky
:Sky rdf:type owl:Class ;
     rdfs:subClassOf :NaturalNonLivingThing .


###  http://example.com/tkgre/mini_vrd_world#TeddyBear
:TeddyBear rdf:type owl:Class ;
           rdfs:subClassOf :Toy .


###  http://example.com/tkgre/mini_vrd_world#Toy
:Toy rdf:type owl:Class ;
     rdfs:subClassOf :EngineeredThing ,
                     :NonBike ,
                     :NonWearableThing .


###  http://example.com/tkgre/mini_vrd_world#Umbrella
:Umbrella rdf:type owl:Class ;
          rdfs:subClassOf :Device .


###  http://example.com/tkgre/mini_vrd_world#UpperBodyClothing
:UpperBodyClothing rdf:type owl:Class ;
                   rdfs:subClassOf :Clothing .


###  http://example.com/tkgre/mini_vrd_world#VRDworldInfrastructure
:VRDworldInfrastructure rdf:type owl:Class .


###  http://example.com/tkgre/mini_vrd_world#VRDworldThing
:VRDworldThing rdf:type owl:Class .
```

```
###  http://example.com/tkgre/mini_vrd_world#Vehicle
:Vehicle rdf:type owl:Class ;
         rdfs:subClassOf :EngineeredThing ,
                         :NonWearableThing ;
         owl:disjointWith :WearCapableThing .


###  http://example.com/tkgre/mini_vrd_world#WearCapableThing
:WearCapableThing rdf:type owl:Class ;
                  owl:equivalentClass [ rdf:type owl:Class ;
                                        owl:unionOf ( :Dog
                                                      :Person
                                                      :TeddyBear
                                                    )
                                      ] ;
                  rdfs:subClassOf :MixedThing ,
                                  :NonFood ;
                  owl:disjointWith :WearableThing .


###  http://example.com/tkgre/mini_vrd_world#WearableThing
:WearableThing rdf:type owl:Class ;
               rdfs:subClassOf :EngineeredThing ,
                               :NonBike .


#################################################################
#    Individuals
#################################################################

###  http://example.com/tkgre/mini_vrd_world#ball100
:ball100 rdf:type owl:NamedIndividual ,
                  :Ball .


###  http://example.com/tkgre/mini_vrd_world#bbox-car101
:bbox-car101 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-car101b
:bbox-car101b rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-dog100
:bbox-dog100 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-hat100
:bbox-hat100 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-hat100b
:bbox-hat100b rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-jeans100
:bbox-jeans100 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-person100
:bbox-person100 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-person101
:bbox-person101 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-shirt100
:bbox-shirt100 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-shoes100
:bbox-shoes100 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bbox-umbrella100
:bbox-umbrella100 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#bike100
:bike100 rdf:type owl:NamedIndividual .
```

```
###  http://example.com/tkgre/mini_vrd_world#car101
:car101 rdf:type owl:NamedIndividual ,
                 :Car ;
        :beside :person101 ;
        :hasBbox :bbox-car101 ,
                 :bbox-car101b .


###  http://example.com/tkgre/mini_vrd_world#cat101
:cat101 rdf:type owl:NamedIndividual ;
        owl:sameAs :cat101b ;
        :eat :pizza101 .


###  http://example.com/tkgre/mini_vrd_world#cat101b

###  http://example.com/tkgre/mini_vrd_world#cat101b
:cat101b rdf:type owl:NamedIndividual ,
                  :Cat .


###  http://example.com/tkgre/mini_vrd_world#dog100
:dog100 rdf:type owl:NamedIndividual ,
                 :Dog ;
        owl:sameAs :dog100b ;
        :beside :bike100 ;
        :hasBbox :bbox-dog100 ;
        :hold :ball100 ;
        :near :bike100 ;
        :nextTo :person100 ;
        :reflexiveProp :person100 ;
        :under :umbrella100 .


###  http://example.com/tkgre/mini_vrd_world#dog100b

###  http://example.com/tkgre/mini_vrd_world#dog100b
:dog100b rdf:type owl:NamedIndividual ,
                  :Dog .


###  http://example.com/tkgre/mini_vrd_world#hat100
:hat100 rdf:type owl:NamedIndividual ,
                 :Hat ;
        :above :shoes100 ;
        :hasBbox :bbox-hat100 ,
                 :bbox-hat100b ;
        :on :person100 .


###  http://example.com/tkgre/mini_vrd_world#hat101
:hat101 rdf:type owl:NamedIndividual ;
        :on :person101 .


###  http://example.com/tkgre/mini_vrd_world#jeans100
:jeans100 rdf:type owl:NamedIndividual ,
                   :Jeans ;
          :hasBbox :bbox-jeans100 .


###  http://example.com/tkgre/mini_vrd_world#person100
:person100 rdf:type owl:NamedIndividual ;
           :below :umbrella100 ;
           :eat :pizza100 ;
           :has :shirt100 ;
           :hasBbox :bbox-person100 ;
           :hold :pizza100 ;
           :ride :bike100 ;
           :wear :hat100 ,
                 :jeans100 ,
                 :shirt100 ,
                 :shoes100 .


###  http://example.com/tkgre/mini_vrd_world#person101
:person101 rdf:type owl:NamedIndividual ,
                    :Person ;
           :hasBbox :bbox-person101 ;
           :hold :cat101 ,
                 :teddybear101 ;
           :irreflexiveProp :teddybear101 ;
           :wear :hat101 .
```

```
###  http://example.com/tkgre/mini_vrd_world#pizza100
:pizza100 rdf:type owl:NamedIndividual ,
                   :Pizza .


###  http://example.com/tkgre/mini_vrd_world#pizza101
:pizza101 rdf:type owl:NamedIndividual .


###  http://example.com/tkgre/mini_vrd_world#shirt100
:shirt100 rdf:type owl:NamedIndividual ,
                   :Shirt ;
          :hasBbox :bbox-shirt100 ;
          :on :person100 .


###  http://example.com/tkgre/mini_vrd_world#shirt101
:shirt101 rdf:type owl:NamedIndividual ,
                   :Shirt .


###  http://example.com/tkgre/mini_vrd_world#shoes100
:shoes100 rdf:type owl:NamedIndividual ,
                   :Shoes ;
          :hasBbox :bbox-shoes100 .


###  http://example.com/tkgre/mini_vrd_world#sky101
:sky101 rdf:type owl:NamedIndividual ,
                 :Sky ;
        :above :car101 .


###  http://example.com/tkgre/mini_vrd_world#teddybear101
:teddybear101 rdf:type owl:NamedIndividual ,
                       :TeddyBear ;
              :nextTo :cat101 ;
              :wear :shirt101 .


###  http://example.com/tkgre/mini_vrd_world#umbrella100
:umbrella100 rdf:type owl:NamedIndividual ,
                      :Umbrella ;
             :above :person100 ;
             :hasBbox :bbox-umbrella100 ;
             :reflexiveProp :person100 .


###  Generated by the OWL API (version 4.5.26.2023-07-17T20:34:13Z) https://github.com/owlcs/owlapi
```