Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

WinoGrammar

Using Answer Set Grammars to Learn Explanations for Winograd Schemas

Author:
Jordan Spooner

Supervisors:
Prof. Alessandra Russo
Dr. Krysia Broda

Second Marker:
Prof. Lucia Specia

June 15, 2020

Abstract

The Winograd Schema Challenge (Levesque et al., 2012) is a set of 273 expert-crafted pronoun resolution problems which form a benchmark for commonsense reasoning. In recent years, several approaches to the WSC using neural language models have significantly pushed forward the state-of-the-art accuracy on the dataset. However, the performance of these approaches on certain subsets of problems with strong statistical hints, as well as their reliance on extensive fine-tuning procedures, has brought into question whether these models are truly demonstrating commonsense reasoning, or whether they are instead exploiting spurious biases present in the dataset. One frequently proposed method to test whether an approach to the WSC demonstrates true commonsense capabilities is to require it to *explain* its answers, but there is currently no published approach that does this.

In this project, we propose an approach to the WSC that uses inductive logic programming to *learn* explanations from similar examples that are sourced automatically. In particular, we make use of answer set grammars to automatically generate structured representations for natural language texts, and propose a fully-automated approach to generating induction tasks which learn commonsense rules directly within these grammars. We are then able to determine the answers to Winograd schemas by applying these learned rules. Our work makes two main contributions. Firstly, we show that answer set grammars can be used to elegantly encode the syntax and semantics of natural language, producing structured representations for texts that are richer than dependency parses, but can be generated without a significant decrease in accuracy compared to a dependency parse alone. Secondly, we demonstrate that our end-to-end approach to the WSC is comparable in accuracy to most previous approaches, but is additionally able to generate natural language explanations to support our answers, which are valid in 85% of cases.

Acknowledgements

I would like to thank Prof. Alessandra Russo and Dr. Krysia Broda for their guidance and support throughout this project, and for all the time they have devoted to our weekly meetings.

Contents

Chapter 1

Introduction

Improving the ability of computational agents to reason with commonsense has been a long-standing challenge in Artificial Intelligence research, with its origins tracing as far back as the 1960s. Such reasoning has far-reaching applications in areas such natural language processing (NLP), vision and robotics, where many problems present innate ambiguities, capable of being resolved only with a rich understanding of the world. Despite this, progress in the area has been frustratingly slow, and even today there are few commercial systems which make any significant use of automated commonsense reasoning (Davis and Marcus, 2015).

In this project, we focus our attention on a subset of particularly difficult coreferencing problems, called *Winograd Schemas*, which are often presented as a modern-day alternative to the Turing Test (Levesque et al., 2012). We combine state of the art approaches from natural language processing, deep learning and logic-based learning in order to develop a novel method for performing the kind of commonsense reasoning required by these problems.

1.1 Motivation

A *Winograd Schema* is a small reading comprehension test with a single binary question, intended to assess a system's ability to perform commonsense reasoning. An example might be:

> The fish ate the worm. **It** was *tasty*. What was *tasty*?
>
> *Answer 0*: the fish
> *Answer 1*: the worm

While most humans have no difficulty understanding that it was *the worm* that was tasty, AI systems have traditionally struggled to determine the correct answer, which requires the machine to both obtain and apply the commonsense knowledge that "something you eat can be tasty". In fact, until recently, state-of-the-art approaches have barely performed better than chance on these kinds of problems, and even today there is no approach that is able to achieve human-level performance.

Historically, approaches to commonsense reasoning problems of this kind have been based heavily on formalizations using mathematical logic. Significant work has been done on developing large, structured commonsense knowledge databases, and methods for automated reasoning over them. Generally these resources have been built by hand (Lenat et al., 1990), crowd-sourcing (Speer et al., 2017), or web-mining (Mitchell et al., 2015), and as such, they often suffer from missing rules and/or inconsistencies.

In recent years, advances in deep learning have enabled new state of the art results for many commonsense reasoning problems, including the *Winograd Schema Challenge* (WSC) (Levesque et al., 2012). Most notably, neural language models, which learn probability distributions over word sequences, when trained over huge text corpora, are innately able to provide some conjecture as to whether a sentence makes sense or not (Trinh and Le, 2018). However, despite their state-of-the-art performance, these approaches also suffer from several limitations. In particular, their performance on certain classes of Winograd schemas seems to suggest that in many cases language models exploit very simple statistical patterns (e.g. *the fish is tasty* is more likely than *the worm is tasty*), and as such they can struggle to solve problems which require a deeper understanding of the context introduced by the sentence (e.g. the fact that *the worm was eaten*). In light of these observations, many have questioned the commonsense reasoning capabilities of language models (Da and Kasai, 2019; Petroni et al., 2019), and advocated for an approach to the WSC which is able to *explain* its answers (Morgenstern and Jr., 2015; Zhang et al., 2020).

In an attempt to develop an explainable approach, we propose combining neural approaches for natural language understanding with a symbolic approach for learning commonsense rules. Our approach is motivated in particular by recent advances in deep learning, which have enabled highly accurate parsers for natural language (Clark et al., 2018; Zhou and Zhao, 2019), and recent research in inductive logic programming, which has demonstrated the capability to learn *answer set grammars* (ASGs) (Law et al., 2019). In brief, ASG induction allows the semantic constraints of a language, in the form of Answer Set Programming (ASP) annotations, to be learned, given the language's syntax and examples of semantically valid and invalid sentences.

In light of this work, we propose a novel approach which learns semantic constraints on-top of an automatically generated grammar for natural language. Such a system would benefit from intrinsic structural knowledge of the text, making the semantics easier to infer, and would learn ASP rules, allowing complex and context-sensitive concepts to be learned. These learned rules would encode commonsense knowledge, that could be applied to Winograd schemas in order to resolve their coreferencing ambiguities, and additionally could be used directly as explanations.

1.2 Objectives

The goal of this project is to explore the applicability of answer set grammars in developing an approach for solving difficult coreferencing problems which require commonsense knowledge. At a high-level, the objectives of this project are to:

1. **Investigate the applicability of ASGs to natural language understanding in general.** In particular, we ask whether it is possible to automatically generate base ASGs capable of representing the structure of natural English language, and whether there is any benefit in doing so. To be applicable to the WSC in particular, the process for generating base ASGs would need to be domain-independent and the generated grammar would need to be able to accurately parse training and test examples. Furthermore, these grammars would need to be annotated with as much semantic information as can be reliably determined, in order to allow complex semantic rules to be learned.

2. **Develop an approach for automated retrieval of relevant examples,** which could then be used to learn the commonsense knowledge required to solve a given coreferencing problem, as per the third objective. This involves developing methods for both searching for and selecting related texts, based on their commonality and relevance to the original text.

3. **Investigate the use of ASG induction for learning commonsense knowledge.** We aim to explore the idea of automatically generating ASG learning tasks capable of extending a base grammar with commonsense rules, given relevant examples. This would require training texts to be automatically translated into positive and negative examples. Also key to this objective is the need to automate the specification of the hypothesis space, which must be large enough to contain the intended rules, but small enough as to be efficiently searchable.

4. **Evaluate an ASG-based approach to coreference-resolution.** Finally, we ask whether the natural language understanding capability of ASGs can be combined with inductive learning of commonsense knowledge and applied in order to solve Winograd schemas. In addition to quantitative evaluation (i.e. what accuracy can be achieved?), a key part of this objective is a qualitative evaluation of the rules learned (i.e. can the system offer sufficient explanation for its choices?), and a critical analysis of the approach compared to the many existing approaches to the WSC.

1.3 Project Overview

Figure 1.1 provides a very brief overview of our proposed approach for learning commonsense knowledge to solve coreferencing problems, which we will discuss in detail in Chapters 5 and 6. We first use deep learning models and existing knowledge bases to generate a *knowledge graph* structure for the problem, which encodes its semantics. This knowledge graph is then used to search for and select semantically similar examples with unambiguous coreferences, in a *knowledge hunting* process. An *answer set grammar* is generated to encode the compositional semantics of the examples, and a *learning task* is generated, in which the learner attempts to extend the ASG with a *hypothesis* that is able to explain the coreferencing in the selected examples. Finally, the resulting ASG is used to parse the problem text, and resolve its ambiguous coreference. Crucially, the resolution made by the approach can be explained in terms of the learned hypothesis.

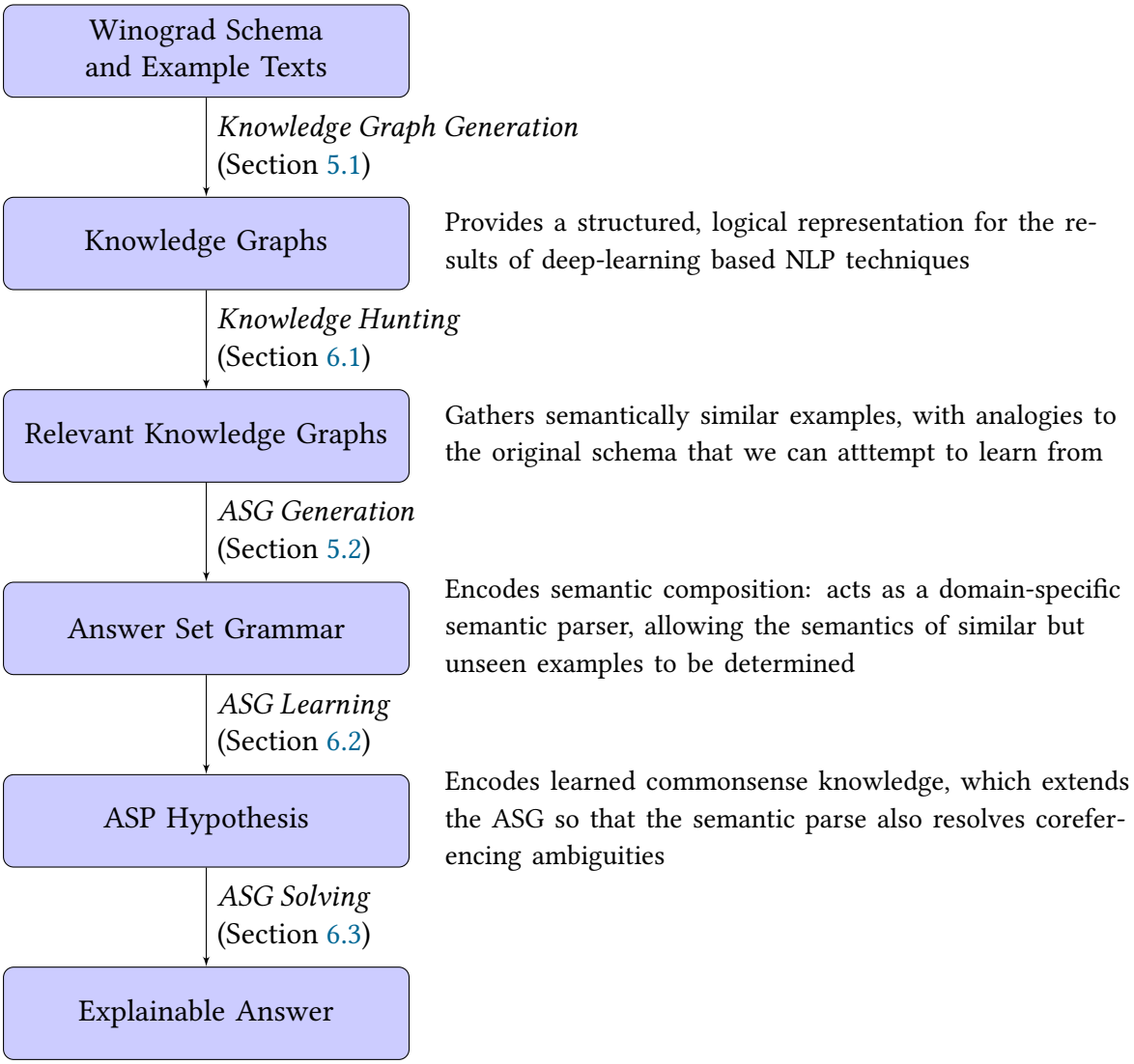


Figure 1.1: An overview of our approach for learning commonsense knowledge to solve coreferencing problems.

1.4 Contributions

Our contributions in this project are threefold:

1. We propose a **formalism for expressing natural language using answer set grammars**, which is linguistically motivated by head-driven phrase structure grammar. It encodes both syntactic and semantic composition in a single structure, and so can both express complex grammatical constraints and be used directly for semantic parsing. We also describe and implement a fully automated and domain-general approach for generating domain-specific ASGs for natural language which are both accurate and semantically-rich. We evaluate our approach on the WSC dataset and show that it encodes significantly richer semantic information than a dependency parse alone, but with an accuracy that is comparable to state-of-the-art dependency parsing approaches.

2. We propose and implement a novel **semantics-guided knowledge hunting approach**, which selects semantically similar knowledge texts which are highly relevant for learning by analogy.

3. We propose an **approach to Winograd-style coreference resolution using the learning from answer set grammars paradigm**, which is based on the goal of *learning* the commonsense knowledge required to solve each problem. We implement a fully-automated system for solving Winograd schemas based on the approach, which to our knowledge is the first such system which supports its answers with explanations. Furthermore, unlike other approaches, our results can be fine-tuned by incorporating existing or hand-engineered background knowledge. We evaluate our approach on the full WSC dataset and show that it is both highly precise and capable of handling schemas that state-of-the-art approaches cannot.

Chapter 2

Datasets

This project focuses mainly on two datasets for natural language understanding: the bAbI tasks dataset and the Winograd Schema Challenge (WSC). The WSC consists of particularly challenging coreferencing problems, and does not provide any training examples. As such, we initially focus on the bAbI tasks dataset, which has a smaller vocabulary and provides a large, noiseless set of training examples. This should make the learning process easier, allowing us to initially attend to the development of a robust and flexible approach to representing natural language understanding problems using answer set grammars.

2.1 bAbI Tasks

The bAbI tasks (Bengio and LeCun, 2016) are "a set of prerequisite toy tasks" for testing text understanding and reasoning. The dataset is made up of 20 different tasks, each testing a single skill. Each task features 1000 training examples and 1000 testing examples, all of which are noiseless. The goal is not only to achieve high accuracy on the test examples, but also to use as few training examples as possible.

In particular, we choose to focus on tasks 11 (*basic coreference*) and 13 (*compound coreference*). Examples from each task are shown below:

Task 11 (*basic coreference*)

John went to the bathroom.	
He then journeyed to the office.	
Where is John?	*A:* office

Task 13 (*compound coreference*)

Sandra and John went to the bedroom.	
Following that they went to the hallway.	
Where is John?	*A:* hallway

2.2 Winograd Schema Challenge

The *Winograd Schema Challenge* (WSC) (Levesque et al., 2012) is commonly presented as an alternative to the Turing Test, a test proposed by Alan Turing in 1950 for deciding whether or not machines are able to think. The idea of Turing's test is that if after a long unrestricted conversation with a machine in natural English, a human interrogator is unable to tell whether they are dealing with a machine or a human, then we should consider this as evidence that the machine is capable of thought. Such a test, however, has conceptual and practical drawbacks, most notably, that machines can merely rely on simple means, built around deception and evasiveness, to maintain a realistic conversation (Weizenbaum, 1966).

The WSC is instead a test made up of a set of 273 *Winograd Schemas*. Each WS is a small reading comprehension test with a single binary question.

Example 2.1. Levesque et al. (2012) give the following example of such a question:

> The town councillors refused to give the angry demonstrators a permit because **they** *feared* violence. Who *feared* violence?
>
> *Answer 0*: the town councillors
> *Answer 1*: the angry demonstrators

Each question must satisfy four conditions:

1. Two parties are mentioned in the text by noun phrases (e.g. "the town councillors" and "the angry demonstrators"). We call these the *candidate referents*.
2. A pronoun or possessive adjective is used to reference one of the parties (e.g. "they/them/their"). Importantly, it must be the right sort for both parties (e.g. both parties must be groups). We call this the *target pronoun*.
3. The question asks the test-taker to determine the referent of the target pronoun, by choosing between the two candidate referents.
4. There is a *special word* in the text, which when replaced by another word (the *alternate word*), forms a valid question, but with the opposite answer.

To explain the fourth condition, consider again Example 2.1. For this example, the special word is *feared*, and the alternate word is *advocated*:

> The town councillors refused to give the angry demonstrators a permit because **they** *advocated* violence. Who *advocated* violence?
>
> *Answer 0*: the town councillors
> *Answer 1*: the angry demonstrators

It is the fourth condition which the authors of the challenge claim requires *thought* to get a correct answer. The contexts of where the special and alternate word may appear tend to be statistically similar, and thus it is argued that the test-taker needs to have background knowledge not expressed in the sentence in order to correctly resolve the reference.

Non-examples Whilst a Winograd Schema should be easily disambiguated by a human reader, there are some examples that are "too easy". One commonly cited example is:

> The women stopped taking the pills because **they** were [*pregnant*/*carcinogenic*]. Which individuals were [*pregnant*/*carcinogenic*]?
>
> *Answer 0*: the women
> *Answer 1*: the pills

The trick with this example is that only the women can be pregnant and only the pills can be carcinogenic, and so there is no need to fully understand the sentence. These restrictions are called *selectional restrictions*, and can easily be learned by considering a large-enough text corpus.

Similarly, consider the example, again from (Levesque et al., 2012):

> The race car zoomed by the school bus because **it** was going so [*fast*/*slow*]. What was going so [*fast*/*slow*]?
>
> *Answer 0*: the race car
> *Answer 1*: the school bus

Although there is no selectional restriction to be applied in this case, clearly a fast race car is much more common than a fast school bus. Such examples are sometimes called *associative*. It is for this reason that Winograd schemas should be *non-associative*, i.e. there should be no statistical properties of the special and alternate words that would allow the correct answer to be selected with large text corpora alone.

Finally, consider the commonly-cited example:

> Frank was *pleased* when Bill said that **he** was the winner of the competition. Who was the winner?
>
> *Answer 0*: Frank
> *Answer 1*: Bill

The issue with this example is instead that it is genuinely ambiguous — Frank could be pleased about his own or Bill's success.

Associativity, One-Way Ambiguity and Switchability Despite best attempts, it can be argued that there are several examples present in the WSC dataset that do not meet these conditions.

An analysis by Trichelair et al. (2019) finds that 37 of WSC's 286 examples actually are *associative*: i.e. the answer is given away by statistical correlation alone.

Furthermore, a human baseline conducted by Glass and Kim (2015) identified several particularly difficult questions. One example is:

I couldn't put the pot on the shelf because **it** was too *tall*. What was too *tall*?

Answer 0: the pot
Answer 1: the shelf

For this question, more than half of the respondents incorrectly answered that it was the shelf that was too tall. We will refer to questions as *one-way ambiguous* if less than 75% of respondents were unable to determine the correct answer. The dataset contains 19 such examples.

Finally, an example is called *switchable* (Trichelair et al., 2019) if the positions of the two candidate referents can be swapped and the example still makes sense. Since switching the referents should trivially switch the question's answer, these examples can be used to determine whether an approach reasons *consistently*. Trichelair et al. (2019) identifies 131 such examples.

Similar Datasets The WSC is a small dataset with no training data. As such, many attempts have been made at producing larger datasets of similar examples, mainly for the purpose of training statistical models. We briefly mention some examples:

1. **The DPR dataset** (Rahman and Ng, 2012) is made up of 1,882 sentence pairs, in the same format as the WSC problem. Users have consistently noted that DPR is easier than the WSC itself, with a much higher proportion of associative examples.

2. **The GLUE-WNLI dataset** (Wang et al., 2018) transforms the existing WSC problems into an entailment problem. An evaluation set consists of new examples derived from fiction books, however these examples have been found to be significantly easier than those in the WSC.

3. **The SuperGLUE-WSC dataset** (Wang et al., 2019) rephrases the GLUE-WNLI problems back into a more-standard coreference resolution task.

4. **The KnowRef dataset** (Emami et al., 2019) consists of 8,724 Winograd-like examples. These examples were scraped from Wikipedia, OpenSubtitles and Reddit comments, and filtered by hand.

5. **The WikiCREM dataset** (Kocijan et al., 2019a) consists of 2,438,897 examples sourced by masking the second occurrence of repeated names in sentences from Wikipedia. It is noted that roughly 20% of the examples are deemed to be unsolvable, since there is no human filtering.

6. **The WinoGrande dataset** (Sakaguchi et al., 2020) consists of 43,972 Winograd-like examples, which are collected and validated via crowdsourcing, and then go through a process of adversarial filtering — essentially reducing bias in the dataset by removing the examples found easiest by current approaches. The final examples appear to be considerably more difficult than WSC problems.

Challenges The most significant challenge of the WSC comes from the diversity of commonsense knowledge and reasoning ability required to answer the questions correctly. Each question requires a different snippet of commonsense knowledge, and human test-takers intuitively use a combination of spatial, temporal, causal, epistemic and other kinds of reasoning to guide themselves to an answer. For this reason the most successful approaches usually avoid direct reasoning, instead favouring a statistical approach.

However, the authors of the challenge note that demonstrating this kind of complex reasoning is precisely the goal (Levesque, 2014). For example, consider the schema:

> The large ball crashed right through the table because **it** was made of [*styrofoam*/*steel*]. What was made of [*styrofoam*/*steel*]?
>
> *Answer 0*: the large ball
> *Answer 1*: the table

Suppose, as Levesque (2014) does, that the special/alternate word was replaced with "XYZZY", and we were told several facts about this XYZZY material, including that it is mostly air. Richard-Bollans et al. (2017) note that a human should still be able to answer this new question, whilst most statistical approaches will simply learn that "things made out of styrofoam are more likely to be crashed through than to crash through things" — a clearly insufficient explanation which does not generalize to the invented XYZZY material. For this reason, Morgenstern and Jr. (2015) point out that the best approaches should be able to *explain* their reasoning.

We discuss existing (statistical and reasoning-based) approaches to the WSC in more depth in the following chapter.

Chapter 3

Related Work

In this chapter, we provide an in-depth analysis of the existing work on the Winograd Schema Challenge. There is a significant amount of published work on the WSC and other related commonsense reasoning problems, with a diverse range of approaches having been proposed. We classify these approaches into five broad categories: logical reasoning (Section 3.1), knowledge hunting (Section 3.2), traditional machine learning (Section 3.3), machine learning with pre-trained language models (Section 3.4) and hybrid (neural and symbolic) approaches (Section 3.5).

3.1 Logical Reasoning

Many approaches have been proposed to the WSC which involve some form of logical reasoning (Schüller, 2014; Bailey et al., 2015; Sharma, 2019). However, the reasoning required to determine the correct candidate referent in many cases is complex, and the commonsense knowledge is specific and thus difficult to source. For this reason, most approaches explicitly provide the required knowledge, and limit their evaluation set to just a small number of Winograd schemas, which do not require complex social, situational and spatio-temporal reasoning and default assumptions.

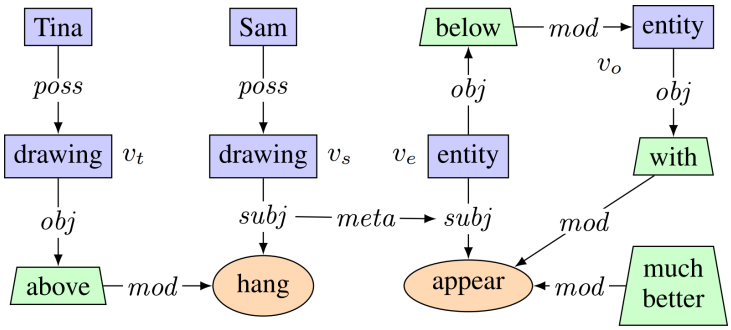


Figure 3.1: Example of the graphical representation proposed by Schüller (2014)

Knowledge Graphs Schüller (2014) defines a kind of knowledge graph, over which reasoning is performed. An example from the paper, corresponding to the sentence "Sam's drawing was hung just above Tina's and it did look much better with another one below it.", is shown in Figure 3.1. The constraints that $v_t \neq v_s$, $v_e \neq v_o$ and $v_t = v_e \vee v_s = v_e$ are also included by the proposed formalism.

Similar graphs are constructed, say, to represent the commonsense knowledge such as that "if v_1 is above v_2, then v_2 is below v_1". The reasoning process then takes the form of two steps. First, isomorphisms are identified between the knowledge graphs, to determine which knowledge is useful for solving the WS. The identified knowledge graphs are then combined by aligning the matching nodes.

Arguably the most significant drawback of this approach is that both the input and background knowledge graphs must be manually constructed, i.e. only the reasoning process itself is automated. While constructing the input graph for each Winograd schema should be possible using state-of-the-art parsing techniques, automating the generation of highly accurate and relevant background knowledge appears to be a much more daunting task, as often this knowledge is incredibly specific.

One challenge faced by the approach is that, in many cases, there may be enough evidence to prove either of the two candidate referents is correct. In these cases, there needs to be a method to determine which answer is more *relevant*. Schüller (2014) identifies features of more relevant explanations:

1. The background knowledge *integrates well* with the input (the schema's knowledge graph). I.e. most of the nodes from the background can be combined with input nodes.
2. The background knowledge forms *contextual implications*, by connecting parts of the input graph that were not originally connected.

This use of *relevance theory* to decide how to integrate background knowledge shows particular promise. For evaluation, schemas with ambiguities were specifically selected. For example, for the sentence "Sam's drawing was hung just above Tina's and it did look much better with another one below it", one could imagine a case where there are three drawings. It is shown that the features described handle the disambiguation process and yield the correct answer for all evaluated schemas.

It is these kinds of default assumptions that present the most significant challenge to logic-based approaches to the WSC. Whether to accept or reject default assumptions requires world knowledge and pragmatic understanding. For example, Richard-Bollans et al. (2017) point out the difficulty of the Winograd schema:

> Tom threw his school bag down to Ray after he reached the [*top*/*bottom*] of the stairs. Who reached the [*top*/*bottom*] of the stairs?
>
> *Answer 0*: Tom
> *Answer 1*: Ray

Here, to derive the correct answer, one must make assumptions regarding the initial position of Tom and Ray (we usually assume that Tom and Ray are initially at the same location),

the movement of the person who reaches the top of the stairs (we assume they have gone up the stairs), and the other person (we assume that they have not moved). Clearly, other assumptions are possible. Furthermore, as noted by Richard-Bollans et al. (2017), it is difficult to formalize the commonsense knowledge required. For example, any knowledge regarding the relation $at\_the\_top\_of$ would need to handle the vagueness of the concept — if one is at the top of a building, are they on the top floor or on the roof?

Automated Approaches Sharma (2019) describes a very similar, but more automated approach, which is able to handle 200 WSC problems, by reasoning with additional knowledge.

A semantic parser is used to generate a graphical form for each Winograd schema. To do this, a dependency parser is used, and then the parsed text is annotated with additional semantic information. Specifically, a rule-based system is used to map syntactic dependencies into semantic dependencies, and discourse connectives are assigned labels (e.g. "and", "but", comma and period are labelled as *next_event*; "because" is labelled as *causes*; and "so" is labelled as *caused_by*). Finally, WordNet (Miller, 1995) is used to annotate each node with classes (e.g. "man" would be assigned the class *person*).

Consider, for example, the schema:

> **Example 3.1.** The man could not lift his son because **he** was so *weak*.

The graphical representation produced by the parser for this schema is shown in Figure 3.2a.

The additional knowledge is provided in a similar format, with additional *is_same_as* labels resolving the reference. For example, for the example above, the additional knowledge would take the form of:

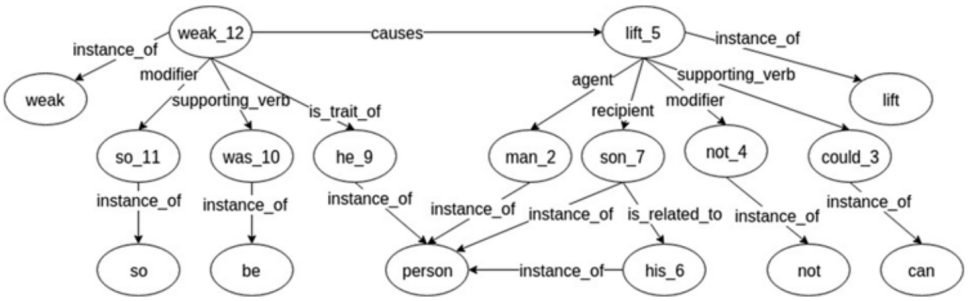
> **Example 3.2.** *IF* person$_1$ cannot lift person$_2$ because person$_3$ is weak *THEN* person$_1$ is the same as person$_3$

The graphical representation of this additional knowledge is shown in Figure 3.2b.

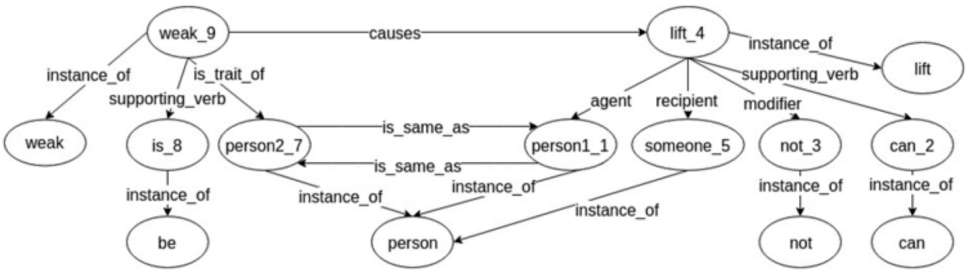
Again, the reasoning process involves aligning the two graphical representations[1]. If the most natural alignment connects the target pronoun with one of the candidate referent with *is_same_as* edges, the answer may then be trivially returned.

The main limitation of this approach, is that the additional knowledge should be of the form shown in Example 3.2. This restriction limits the approach to just 240 of the WSC problems, since 26 problems require multiple pieces of knowledge, and a further 25 require the knowledge that one scenario is *more likely* than another. Additionally, the parser fails to parse a

[1] As noted by Sharma (2019), this is an instance of the NP-complete graph-subgraph isomorphism problem. It can be solved efficiently for small problem sizes using an answer set solver such as `clingo`.



(a) Representation for the schema shown in Example 3.1.



(b) Representation for the additional knowledge shown in Example 3.2.

Figure 3.2: Examples of the graphical representation generated by the semantic parser, *K-Parser* (Sharma, 2019)

further 40 problems, due to errors in syntactic dependency parsing and part-of-speech tagging. For these problems, the graphical representation had to be derived by hand.

We also note that the proposed form of knowledge is both rather unintuitive (presumably not many people attempting to resolve the reference in Example 3.1 would provide the knowledge of Example 3.2 as their justification) and specific (and as such especially difficult to generate). For example, it may often be possible to answer commonsense knowledge questions with knowledge in this format, but far more natural to use multiple pieces of knowledge. Consider Levesque's commonly-cited question "Can a crocodile run a steeplechase?". In this case, the specific knowledge that crocodiles cannot run steeplechases is sufficient, but clearly not as natural as combining the more general knowledge that crocodiles have short legs, animals with short legs cannot jump high, and that jumping is a requirement of a steeplechase.

Sharma (2019) also briefly explored the idea of *knowledge hunting*, with the goal of generating the additional knowledge. They find that the knowledge could be generated in 120 cases, and the correct answer was returned in every case where the knowledge could be found. We discuss knowledge hunting approaches in more detail in the following section.

3.2 Knowledge Hunting

In an attempt to automate the extraction of common sense knowledge, many approaches propose *knowledge hunting* methods which make web searches, or searches through large text corpora, in an attempt to find evidence that supports one candidate referent over another.

Sharma et al. (2015), for example, describe an approach which combines semantic parsing of the problem text with a knowledge hunting module. In their approach, they focus on two specific categories of schema, which cover 71 of the 286 schemas making up the WSC:

1. **Event-event causality.** This category applies to examples in which the candidate referent participates in one event, and the target pronoun in another. The required commonsense knowledge is that one event causes the other. E.g. the sentence "Sid explained his theory to Mark but he could not convince him" would be in this category, since it can be solved with the knowledge that "if X explained s to Y but Z could not convince, then $Z = X$".

2. **Causal attributive.** This category applies to examples which can be solved with the knowledge that an event is causally related to some attribute. E.g. the sentence "Pete envies Martin because he is successful" can be solved using the knowledge that "X envies Y because Y is successful".

We now describe the process of *knowledge hunting* itself. Consider again Example 3.1. First this sentence would be parsed, and then a set of queries are generated. This is done by substituting all nominal entities and certain stop words with wildcards, in this case generating the sentence ".*could not lift.*because.*weak.*". Additional queries are created by replacing the verbs in the query with their respective synonyms, for example, generating ".*could not raise.*because.*weak.*". These queries are inputted to the Google Search API, which acts here as a huge text corpus. As an example, Google may return the sentence:

> She could not lift it because she is a weak girl.

We will refer to such a sentence as a *knowledge text*. Importantly, the references in this knowledge text are unambiguous.

Both the WS and the knowledge text are then parsed by a semantic parser. An example of the generated parse tree, corresponding to the schema in the example, is shown in Figure 3.2a. Solutions are again found by identifying graph-subgraph isomorphisms between the parsed representations of the WS and the knowledge text.

This approach was able to determine an answer for 53 of the 71 schemas which fit the investigated categories, and out of those 53 answers, 49 were correct. Incorrect answers were returned in four cases because the knowledge text was inappropriate, Sharma et al. (2015) found that the schema

> Bob paid for Charlie's college education, **he** is very *grateful*.

was answered incorrectly as their system found the knowledge text "I paid the price for my stupidity. How grateful I am.".

Generalized Knowledge Hunting Emami et al. (2018) propose a more general knowledge hunting method. Their approach consists of four steps:

1. The schema is partially parsed. In particular, the following components are extracted based on a very shallow syntactic parse:

C_1, C_2	the candidate referents
V_C	the context predicate (e.g. verb before the discourse connective)
$Conn$	the discourse connective
P	the anaphor
V_Q	the query predicate (e.g. verb after the discourse connective)

   For example, for the problem "The trophy doesn't fit into the brown suitcase because **it** is too *large*.", we have:

   $$\begin{aligned} C_1 &= \text{the trophy} \\ C_2 &= \text{the suitcase} \\ V_C &= \text{doesn't fit into} \\ Conn &= \text{because} \\ P &= \text{it} \\ V_Q &= \text{is too large} \end{aligned}$$

2. Queries are generated of the form

   $$+Term_C + Term_Q - \text{"}Winograd\text{"}$$

   where $Term_C$ and $Term_Q$ are derived from V_C and V_Q. Specifically, WordNet is used to find synonyms for each, and each set is filtered to those elements with a lexical similarity score (also from WordNet) to the original term exceeding some threshold. The threshold is derived from performance on the DPR dataset.

3. The search results are partially parsed, and filtered to those matching one of the following patterns

   $$\begin{aligned} C'_1\ Pred'_C\ C'_2\ Conn\ C'_3\ Pred'_Q \\ C'_1\ Pred'_C\ C'_2\ Conn\ Pred'_Q\ C'_3 \\ C'_1\ Pred'_C\ Conn\ C'_3\ Pred'_Q \\ C'_1\ Pred'_C\ Conn\ Pred'_Q\ C'_3 \end{aligned}$$

   It is hoped that the references in this sentence are unambiguous — either they repeat the noun phrase explicitly, or the reference can be determined using gender/plurality alone. Then, if C'_3 refers to C'_1, this is treated as evidence that the target pronoun P references C_1, otherwise if C'_3 refers to C'_2, this is seen as evidence that P references C_2. There are two important exceptions:

(a) If the event occurs in the passive voice (e.g. "was called"), then the reference must be swapped.

(b) In some cases the verb may be used by transitively and intransitively (e.g. "he opened the door" vs "the door opened"). Such verbs are called *causatively alternating*. This may also cause the order of the referents to be swapped.

4. Each example is weighted. Specifically, Emami et al. (2018) propose a weight of

$$LenScore(e) + OrderScore(e)$$

where

$$LenScore(e) = \begin{cases} 2 & \text{if } len(Term_Q) > 1 \text{ or } len(Term_C) > 1 \\ 1 & \text{otherwise} \end{cases}$$

$$OrderScore(e) = \begin{cases} 2 & \text{if } Term_C \text{ precedes } Term_Q \\ 1 & \text{otherwise} \end{cases}$$

The weights of examples supporting each candidate referent are summed, and the referent with the greater value is returned as the answer.

This system was able to find valid knowledge texts, and hence determine an answer for 199 problems. Of these problems, 119 (60%) were answered correctly.

To understand the performance of the knowledge hunting process itself, the referents of 876 knowledge texts were manually labelled. Of these 876 texts, 703 (81%) were found to be labelled correctly by the knowledge hunting system. Of the 173 incorrectly-labelled texts, 110 were found to have insufficient evidence for either conclusion. This is one of the major challenges for knowledge-hunting based approaches. Language is used in an efficient way and commonsense knowledge is often left implicit. Richard-Bollans et al. (2017), for example, provide the example "Sam chopped down the tree". Here there is a default assumption that chopping is done with an axe. Often extra information is added only in the unusual cases where the default assumption is incorrect, e.g. "Sam chopped down the tree with a sword".

3.3 Machine Learning

Formalizing the kind of reasoning required to tackle the WSC is notoriously difficult, and so naturally many of the earliest approaches to the WSC instead opt for simple machine learning-based approaches. For example, Rahman and Ng (2012) propose training a ranking support vector machine (SVM), using the DPR dataset. Specifically, each example is transformed into an ordered pair of feature vectors, one for each of the candidate referents, and the ranker is trained with the goal of assigning the vector corresponding to the correct referent in each pair a higher rank.

Their approach uses linguistic features or six kinds:

1. **Narrative chains.** Events are extracted from the text and compared against a large set of narrative chains. Narrative chains (Chambers and Jurafsky, 2009) are composed of a sequence of temporally-ordered events that tend to occur together. One example might be:

   $$\text{arrested(obj)} \rightarrow \text{charged(obj)} \rightarrow \text{convicted(obj)} \rightarrow \text{sentenced(obj)}$$

   where "obj" indicates that the *object* of the verb is common across the sequence.

2. **Semantic compatibility.** This feature extracts from the sentence the two candidate referents (C_1 and C_2), the verb applied to the target pronoun (V), and the words following that verb (W), as well as the adjective (J) if V is some conjugation of the verb to-be. It then generates up to six Google queries, whose result counts are compared against each other as features. For example, for the sentence "the fish ate the worm because it was hungry", the following queries would be generated:

   (a) C_1V: "fish was".

   (b) C_2V: "worm was".

   (c) C_1VW: "fish was hungry".

   (d) C_2VW: "worm was hungry".

   (e) JC_1: "hungry fish".

   (f) JC_2: "hungry worm".

3. **FrameNet.** In the case, for example, where C_1 and C_2 are names, clearly the Google-based features are not particularly helpful. In this case, FrameNet (Ruppenhofer et al., 2006) is used to determine the roles of C_1 and C_2, and the Google queries are generated instead using these roles. For example, for the sentence "John killed Jim, so he was arrested", C_1 (John) has the role "killer" and C_2 (Jim) has the role "victim".

4. **Polarity.** Sentiment analysis is applied to the two candidate referents and the target pronoun, and their values are compared. Consider, for example, the sentence "John was defeated by Jim in the election even though he is more popular". Here "John" is assigned a negative sentiment (having been defeated), "Jim" is assigned a positive sentiment and "he" is assigned a positive sentiment (being more popular). In this case, the discourse connective "even though" would suggest that the polarity should be reversed, and hence the target pronoun is resolved to "John".

5. **Connective-based relations.** A triple of the from $(V, Conn, X)$ is produced, where V is the verb from the first clause, $Conn$ is a discourse connective and X is a verb or adjective in the following clause. For example, "Google bought Motorola because they are rich" generates the triple $(\text{buy}, \text{because}, \text{rich})$. This triple is then checked against a large set of relations extracted from large text corpora.

6. **Lexical features.** These features count single word occurrences, word pairs separated by a discourse connective, and word pairs relating to each candidate referent.

The system was evaluated only on the DPR dataset, and not on the harder WSC dataset, but was the first system to achieve significantly greater-than-chance performance (73% accuracy) on these kind of difficult coreference problems.

Feature analysis demonstrated that their Google-based semantic compatibility features were among the most important. Since WSC problems are designed to be "Google-proof", this likely suggests deficiencies in the DPR dataset — namely that many examples are associative, and hence we expect that the system might not perform nearly as well on the WSC itself.

Furthermore, the achieved accuracy of 73% on DPR is still significantly below that of human performance and more recent approaches, demonstrating that while the features discussed may be helpful, they alone are not sufficient to fully capture the semantics of the problems. Richard-Bollans et al. (2017), for example, note that the FrameNet-based features do not take into account the discourse connective, and thus would be identical for the sentences "John killed Jim, *so* he was arrested" and "John killed Jim *after* he was arrested". Other features suffer from similar deficiencies.

3.4 Language Models

Most recent approaches to the WSC rely on high-capacity language models, trained on a large amount of unlabelled data. They are usually optimised to predict the next token in a sequence, and appear to efficiently store a vast amount of linguistic knowledge, which has been shown to be useful for several downstream tasks.

Recurrent Language Models Trinh and Le (2018) outline a remarkably simple and direct approach to the WSC using recurrent language models. First, the pronoun in the original sentence is swapped out with each of the candidate referents. For example, the schema:

> The trophy doesn't fit into the brown suitcase because **it** is too *large*.
> What is too *large*?
>
> ---
>
> *Answer 0*: the trophy
> *Answer 1*: the suitcase

is translated into two candidate sentences:

> **Example 3.3.** The trophy doesn't fit into the brown suitcase because **the trophy** is too *large*.

and

> **Example 3.4.** The trophy doesn't fit into the brown suitcase because **the suitcase** is too *large*.

A recurrent language model, which predicts a distribution over a large vocabulary of words, conditioned on the preceding words, is trained on large text corpora. Each of the candidate sentences is then scored based on its probability, and the referent occurring in the sentence with the higher probability is selected as the answer.

Trinh and Le (2018) also propose several of minor enhancements to the system to improve its accuracy:

1. A key limitation of the approach is that measuring the sentence's probability as the joint probability of each word in the sentence occurring at its respective position may result in a bias due to a difference in the commonality of each of the candidate referents. For example, "the trophy" is much rarer in text corpora than "the suitcase", which in this case would result in an incorrect preference for candidate sentence 3.4. In order to resolve this issue, a *partial scoring* method is proposed, in which the probability of the words following the candidate referent, given the preceding words, is calculated. E.g. the score for candidate sentence 3.3 under this approach would be

   $$P(\text{is, too, large}|\text{the, trophy, doesn't, fit, into, the, brown, suitcase, because, the, trophy})$$

2. They develop a customized training dataset, based on questions in commonsense reasoning tasks — specifically this corpus is made up of the documents from their selected corpora which contain the most overlapping n-grams with the Winograd schemas.

3. An ensemble of 14 language models, trained on several different text corpora, is used to make a final decision.

This approach, with the adjustments described, achieves a 63.7% accuracy on the WSC. Furthermore, Trinh and Le (2018) note that their language model was able to correctly identify the *special word* for 64.6% of the correctly-answered questions — in these cases it was the word that made the greatest contribution to the difference in probabilities between the two candidate sentences. They claim that this "indicates a good grasp of commonsense knowledge".

However, the approach has been noted to have several flaws. Statistical language models, especially the kind used by Trinh and Le (2018), have been shown to have difficulty learning complex concepts, particularly those which are context-sensitive. Saba (2018) notes, for example, that the statistical approach does not easily capture type similarities, and so claim that the approach needs to many possible combinations individually, instead of a single generalised concept. For example, the trophy, the ball, and the laptop all co-occur in quite different ways, and so each of these concepts might appear markedly different to a recurrent language model:

> The trophy doesn't fit into the brown suitcase because **the trophy** is too *large*.
> The ball doesn't fit into the brown suitcase because **the ball** is too *large*.
> The laptop doesn't fit into the brown suitcase because **the laptop** is too *large*.

This might explain why such a huge volume of textual data is required by the language model to make good predictions, and why a specially-constructed dataset with overlapping n-grams was found to be necessary.

Transformer-Based Language Models In the last couple of years, transformer-based language models have significantly advanced state of the art results on many natural language tasks. Examples include OpenAI's GPT and subsequently GPT-2 models (Radford et al., 2019), which use deep (up to 48-layer) transformer architectures (Vaswani et al., 2017) and huge amounts of training data (over 8 million documents). These language models have also shown significant promise in solving commonsense reasoning problems. For example, GPT-2 has been applied to the WSC, using the same approach as in (Trinh and Le, 2018), but achieving a significantly improved accuracy of 70.7% (Radford et al., 2019).

With such large volumes of training data, an obvious question is whether the corpora include the exact problems in the WSC. Radford et al. (2019) evaluated their training set to find only 10 schemas with significant overlaps and only 1 schema present in its entirety.

Bidirectional Transformers While GPT and GPT-2 are *auto-regressive* models (i.e. they are trained to predict the next token, given knowledge of only previous tokens, similar to in a recurrent language model), Google's BERT (Devlin et al., 2019) takes a different approach. It pre-trains *bidirectional* transformers, using context from both sides of the token to be predicted. For example, a training example might take the form of "Mary bought `[MASK]` for the film", where the goal is to predict the original value of the masked token. Klein and Nabi (2019) show that BERT is able to achieve an accuracy 60.3% on the WSC using the out-of-the-box BERT-base model, without any customized training data or fine-tuning.

Fine-Tuning A common approach to many NLP tasks in recent years has been to apply transfer learning. First, a language model such as BERT or RoBERTa is pre-trained on vast amounts of textual data, using training tasks such as the masked token prediction method discussed earlier. Then, starting from the pre-trained model's weights, a task-specific model is developed through a process of *fine-tuning*, during which the model is trained on tasks much closer to its intended usage.

Ruan et al. (2019) first investigated fine tuning approaches for the WSC. They use the DPR dataset (Rahman and Ng, 2012), consisting of 1886 Winograd-like examples, to fine-tune the BERT-large model, achieving 71.1% accuracy. They find that fine-tuning is particularly important, providing a 10-20% improvement in accuracy, presumably by combining the knowledge from Winograd annotation with the extremely diverse linguistic knowledge present in the out-of-the-box BERT models.

Several approaches have since advanced the state of the art with small-tweaks to the fine-tuning process:

- Kocijan et al. (2019b) use their masked Wikipedia dataset, WikiCREM, for fine tuning, achieving 72.5% accuracy. Their fine-tuning process is slightly optimised: the pronoun to be resolved is masked out of the sentence and the language model is used to predict the correct candidate referent out of C_1 and C_2. This approach has consistently been used to achieve state-of-the-art results on the similar GLUE WNLI and SuperGLUE WSC tasks.

- Ye et al. (2019) used ConceptNet to generate a commonsense focused fine tuning task. Their process finds triples from ConceptNet, aligns the triples with sentences from the English Wikipedia dataset, and masks one token from the ConceptNet relation. The task involves selecting the original value of the masked token. This approach achieves 75.5% accuracy, demonstrating its improved commonsense reasoning capability.
- Sakaguchi et al. (2020) used their large crowd-sourced Winograd-like dataset, WinoGrande, in their fine tuning process, achieving state-of-the-art accuracy of 90.1% on the WSC. However they note that the results "may need to be taken with a grain of salt", as their work suggests that the WSC may contain dataset-specific biases, causing their models to be "right for the wrong reasons" and hence "running the risk of overestimating the true capabilities of machine intelligence on common sense reasoning". The significant advancement in performance can be attributed to the volume of commonsense knowledge present in their large, Winograd-like dataset. In particular we note that their dataset contains many schemas which are extremely similar to those present in the WSC dataset, often with just different noun-phrases, or a small grammatical twist.

Embedding Sentence Structure Ruan et al. (2019) note that the WSC, with its limited training data and sensitivity to sentence structure, is well-suited to approaches which explicitly incorporate dependency structure. They generate a dependency tree for each sentence and modify BERT so that each word attends only to its parent, its children and itself, rather than all words in the sentence. They find that structural information is important, consistently improving accuracy by ~3%, but especially helps for the tricker, non-associative problems, which language models traditionally do poorly on.

Language Models as Knowledge Bases Petroni et al. (2019) provide an explanation as to why language models perform so well on commonsense reasoning tasks, showing that pretrained language models such as BERT capture accurate relational knowledge. They explore the idea by building a simple sentence and using a language model to predict a masked token. For example, to generate the relational concept "CapableOf", one might build a sentence such as:

Raven can `[MASK]`.

for which BERT will predict the tokens "fly", "fight", and "kill", in descending order of probability.

In fact, the out-of-the-box BERT-large model was found to consistently generate "surprisingly reasonable and syntactically correct" commonsense knowledge, often comparable to crowd-sourced rules present in ConceptNet (Speer et al., 2017).

However, since some abstract concepts are uncommon in BERT's training data (largely composed of English Wikipedia articles), and many obvious concepts are often left implicit in text, BERT does struggle on certain examples. One example from (Petroni et al., 2019) might be

A pond is for `[MASK]`.

for which BERT predicts "swimming", whilst ConceptNet proposes "fish". Da and Kasai (2019) also find that fine-tuning BERT with commonsense knowledge embeddings from ConceptNet, WebChild and ATOMIC does improve its performance on commonsense reasoning tasks,

which seems to suggest that while language models do encode some commonsense rules, they do not completely cover the examples present in large relational knowledge bases.

Language Models for Commonsense Reasoning Although language models do seem to encode some commonsense knowledge, they exhibit some surprising behaviour on the WSC, which seems to suggest that they are not truly applying commonsense. Firstly, Trichelair et al. (2019) find that while some language models are able to achieve over 90% on associative problems (those with strong statistical hints), they barely perform better than chance on harder, non-associative problems. Secondly, they find that when the two candidate referents are swapped for switchable problems, most language model-based approaches swap their answer accordingly less than half of the time. This poor performance on non-associative examples and generally inconsistent reasoning seems to suggest that, in many cases, the language models learn only basic lexical patterns, and do not apply general commonsense knowledge as well as one might hope.

3.5 Hybrid Approaches

Prakash et al. (2019) describe an approach which combines knowledge hunting and language model based methods.

The knowledge extraction approach is very close to what is described by (Sharma et al., 2015). From the results, text snippets with less than 30 words are considered, and the 10 snippets most similar to the WSC sentences are kept. Snippets containing 80% or more words from the WSC problem are excluded.

This is followed by an entity alignment step — which attempts to align the target pronoun and each of the candidate referents from the WSC problem with an entity in the knowledge text. The entities are said to align if they are arguments of the same verb (or a synonym of that verb) and have the same semantic role (e.g. agent or recipient) with respect to that verb. Consider, for example, the problem

> *The trophy* doesn't fit into *the brown suitcase* because *it* is too large.

and the knowledge text

> *The CPU fan* would not fit in because *the fan* was too large.

Here, *the trophy* aligns with *the CPU fan*, *it* aligns with *the fan*, and *the brown suitcase* does not align with any entity of the knowledge text. Thus the two facts are derived:

$$aligned\_with(trophy, fan)$$
$$aligned\_with(it, fan)$$

The reasoning approach then uses probabilistic soft logic (PSL). Logical atoms in PSL have soft truth values in the interval $[0, 1]$. This makes PSL well-suited to determining the *most*

probable explanation, given some evidence. In particular, this approach relies on a PSL rule of the form:

$$w : aligned\_with(c, e1) \wedge aligned\_with(p, e2) \wedge similar(e1, e2) \rightarrow coref(p, a)$$

where w is the weight of the rule, c is a candidate referent, p is the target pronoun, and $e1$ and $e2$ are entities (noun phrases) from the knowledge text. The $similar$ predicate is used to represent how similar the two entities in the knowledge text are to each other; its value is determined by BERT. In this example, the truth value of $similar(fan, fan)$ should be 1 since the entities are identical.

Finally, the language model is also used to generate two more grounded $coref$ atoms, corresponding to the two possible answers. The truth value of each grounding is simply the probability for the answer, according to the BERT language model.

The final conclusion is determined by which grounding of the $coref$ atom has the higher truth value. As such, this approach relies on a weighted combination of the kind of statistical language model-based reasoning introduced by Trinh and Le (2018) and the kind of knowledge-hunting reasoning introduced by Sharma et al. (2015).

Such an approach benefits from the high precision demonstrated by knowledge-hunting approaches, especially for non-associative problems, which are difficult for language models. It also benefits from a recall improvement over pure knowledge-hunting methods, since the language model's prediction is used to come up with an answer, even when no suitable knowledge texts can be found. Overall, it achieves 71% accuracy on the full WSC dataset, representing a significant improvement over knowledge hunting or pre-trained language model-based approaches alone.

However, it is also important to note that the approach also suffers from problems associated with both approaches: the language model requires a huge volume of textual data and an expensive training process, and the knowledge hunting approach still occasionally invalidates the correct conclusion by choice of unsuitable knowledge texts. Furthermore, the system's statistical reasoning approach means that it cannot explain the reasoning behind its answers.

3.6 Discussion

When evaluating the different approaches to the WSC, we identify three main criteria: how *accurate* the approach is, what *data* it relies on, and whether it demonstrates real commonsense *understanding*. These criteria are discussed in detail below, and a summary is presented in Table 3.1.

Accuracy The most accurate approaches to the WSC tend to be direct reasoning or knowledge hunting approaches, since these approaches tend to focus on a small subset of problems that they are able to answer with near-perfect accuracy. However, clearly a trade-off is being made by these approaches to favour *precision* over *recall*, as they often choose not to answer questions for which parsing is difficult or no relevant knowledge can be found.

When considering the full WSC, fine-tuned language models have consistently achieved state-of-the-art accuracies in recent years. Hybrid approaches have also been shown to be capable of addressing the poor recall of pure knowledge hunting approaches.

Meanwhile, language models without fine-tuning and other simpler machine learning approaches have significantly trailed behind in terms of accuracy.

Data Approaches that use less training data and are able to generalize quickly should clearly be preferred over those that require tens or hundreds of thousands of examples before they demonstrate any commonsense reasoning capabilities.

Of particular interest are entirely unsupervised approaches, or approaches which require particularly few or no labelled training examples. Knowledge hunting approaches have a clear advantage in this respect, often relying on ten or less example sentences per problem, which are discovered on-the-fly. Language models without fine tuning are also attractive since the learning is entirely unsupervised. However, the kinds of language models used in approaches to the WSC generally have been trained on tens of gigabytes of cleaned textual data, in a process than can take several hours on clusters of GPUs.

Simpler, feature-based, machine learning approaches and fine-tuned language models are even less attractive in this respect, requiring thousands or tens of thousands of labelled, Winograd-like example problems, which are difficult and expensive to source, and mean that these approaches are unlikely to generalize as well to other tasks.

Another issue is whether the training data is too similar to the WSC problems, meaning the actual learning required of the models is more limited than one might expect. In particular, fine-tuned language models are likely to see problems very similar to those in the WSC, multiple times during their training process.

Understanding This leads on naturally to the question of whether the approaches actually demonstrate commonsense reasoning, or are simply learning shallow statistical rules. This can be difficult to determine, especially in the case of language model-based approaches. Although language models have been shown to capture some commonsense *knowledge*, their poor performance on non-associative problems and often inconsistent answers seems to suggest that their commonsense *reasoning* capabilities are limited.

One way to prove that a system is doing real commonsense reasoning is by asking it to explain its answers. None of the approaches we have seen, except perhaps those using direct logic-based reasoning approaches, would be able to do this.

Approach	Advantages	Limitations
Logical reasoning (Schüller, 2014; Bailey et al., 2015; Sharma, 2019)	+ High precision + Consistent + Explainable	- Requires manually constructed knowledge - Poor recall
Knowledge hunting (Sharma et al., 2015; Emami et al., 2018)	+ Often high precision + Uses unlabelled examples, discovered on-the-fly	- Poor recall due to difficulty of finding relevant knowledge - Inconsistencies from contradictory knowledge texts - Cannot explain answers
Machine learning (feature-based) (Peng et al., 2015; Liu et al., 2017)	+ No expensive pre-training	- Poor accuracy compared to LMs - Requires labelled training data - Cannot explain answers
Language models (no fine-tuning) (Trinh and Le, 2018; Radford et al., 2019; Klein and Nabi, 2019)	+ Unsupervised approach	- Poor accuracy compared to fine tuning - Expensive pre-training - Answers inconsistently - Cannot explain answers
Language models (fine-tuning) (Ruan et al., 2019; Kocijan et al., 2019b; Ye et al., 2019; Sakaguchi et al., 2020)	+ State-of-the-art accuracy	- Expensive pre-training - Requires labelled training data - Answers inconsistently - Cannot explain answers
Hybrid (knowledge hunting and LM) (Prakash et al., 2019)	+ Unsupervised approach	- Expensive pre-training - Cannot explain answers

Table 3.1: Comparison of selected approaches to the WSC

Chapter 4

Background

4.1 Natural Language Processing

A key requirement of this project is to be able to accurately parse both Winograd schemas and example sentences, in order to construct answer set grammars that are able to represent them. In this section, we provide a brief overview of modern approaches for processing, parsing and representing natural language.

4.1.1 Preprocessing Methods

Before almost any natural language processing task can be performed on a body of text, the raw text must be normalised into a consistent format. Usually this involves at least: tokenisation of individual words and punctuation, normalisation of word formats and segmentation of sentences. More complex tasks, however, might require further pre-processing such as part-of-speech tagging, named entity recognition, and word sense disambiguation.

Tokenisation and Sentence Splitting Tokenisation is the process of splitting text into a sequence of *tokens*, roughly corresponding to individual words or punctuation. This is mostly a simple process, with the only major hurdle being the need to distinguish between punctuation internal to a word, from the punctuation determining the sentence structure. For example, abbreviations ("Ph.D"), dates ("01.01.2020") and numbers ("100,000"), are generally assigned a single token. Meanwhile, *clitic contractions*, such as "doesn't" are often split into two tokens, "does" and "n't" (Jurafsky and Martin, 2009).

A related problem is that of sentence segmentation, which is important to many NLP tasks, such as parsing, which usually operate on single sentences at-a-time.

Traditionally, tokenisation and sentence segmentation have been implemented by deterministic algorithms based on carefully-crafted regular expressions, which are compiled into efficient finite state automata. However in recent years, neural approaches, often using bidirectional long short-term memory networks (BiLSTMs) have become more common (Qi et al., 2018).

Part-of-Speech Tagging *Part-of-speech tags* (POS tags) classify words according to their role in a sentence. For example, common POS tags include `DT` (determiner, e.g. "the"), `NN` (singular noun, e.g. "worm") and `VBD` (verb in the past tense, e.g. "ate"). The role of a *POS tagger* is to assign each token one of these tags.

Modern POS taggers achieve accuracies in the range of 97-98%. State-of-the-art approaches generally take into account context from both sides of the word being tagged, for example by using BiLSTMs (Qi et al., 2018).

Lemmatisation *Lemmatisation* is the task of identifying the lemma for each word in a text. The *lemma* is the base or dictionary form for a word. For example, the words "am", "are" and "is" all share the lemma "be". The process must take context into account, so for example, the word "saw" may be assigned the lemma "see" or "saw" depending on if it is used in the place of a verb or a noun (Jurafsky and Martin, 2009). Many approaches to lemmatisation use a simple lookup table from word/POS pairs to their lemmatised forms (Qi et al., 2018).

Named Entity Recognition Another related problem is that of *named entity recognition* (NER), in which the goal is to find the proper names in a text and label each with a type. For example, a named entity might be assigned the type *person*, *place*, *organisation*, etc.

NER is often treated as a word-by-word sequence labelling task, in which the tags state the type of each token. As such, modern approaches generally make use of sequence classifiers such as a BiLSTMs or transformers (Jurafsky and Martin, 2009).

4.1.2 Syntactic Parsing

4.1.2.1 Constituency Parsing

Constituency parsing is a task that involves assigning a syntactic (phrase) structure to a sentence, by producing a parse tree that groups words together into *constituents*. Typically, these parse trees serve as an intermediate representation in the process of semantic analysis (Jurafsky and Martin, 2009).

Context Free Grammar Most commonly, constituency parsing builds the kind of structures assigned by a context-free grammar (CFG) (Hopcroft et al., 2007).

Definition 4.1. (Context free grammar) A context free grammar consists of:

- A set of *terminal symbols* (corresponding to the vocabulary of the grammar),
- A separate set of *non-terminal symbols* (which express abstractions over the terminal symbols),
- A set of *production rules* (which express how a given non-terminal symbol can be rewritten as a sequence of terminal and non-terminal symbols), and

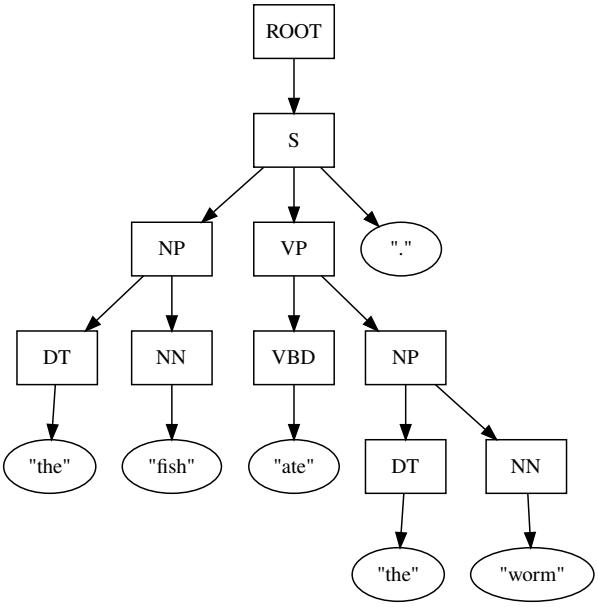


Figure 4.1: A constituency parse tree for the sentence "the fish ate the worm." according to the context free grammar defined in Example 4.1. Terminal nodes are shown by ellipses, non-terminal nodes are shown by rectangles.

- A distinguished non-terminal symbol called the *start symbol*.

A grammar is said to *accept* a string, if it is possible, starting from the start symbol, to use the production rules to generate that string; i.e. it is possible to produce a parse tree for the string.

Example 4.1. (Context free grammar) Consider a CFG with the following production rules:

```
ROOT -> S
S -> NP VP "."
NP -> DT NN
VP -> VBD NP
DT -> "the"
NN -> "fish"
VBD -> "ate"
NN -> "worm"
```

In this example, the terminal symbols are $\{\texttt{the}, \texttt{fish}, \texttt{ate}, \texttt{worm}, .\}$, and the non-terminal symbols are $\{\texttt{ROOT}, \texttt{S}, \texttt{NP}, \texttt{VP}, \texttt{DT}, \texttt{VBD}, \texttt{NN}\}$, which represent different kinds of constituents. `NP` and `VP`, for example, refer to constituents of the type "noun phrase" and "verb phrase" respectively. The start symbol is `ROOT`. As an example, this grammar accepts the string "the fish ate the worm", as demonstrated by the parse tree in Figure 4.1. The process of finding such parse trees given a grammar and a sentence can be done be efficiently, for example by using the CKY algorithm (Jurafsky and Martin, 2009).

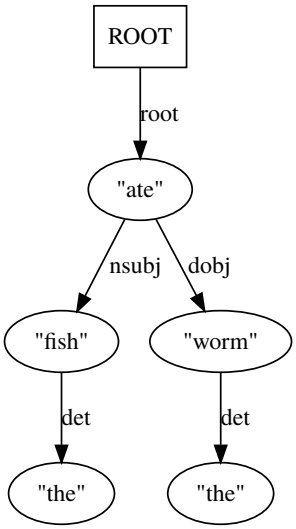


Figure 4.2: A dependency parse tree for the sentence “the fish ate the worm.”

4.1.2.2 Dependency Parsing

Dependency grammars are a family of grammar formalisms that instead describe the structure of sentences in terms of just their words and a set of grammatical relations (Jurafsky and Martin, 2009). For example, Figure 4.2 shows the dependency tree for the sentence “the fish ate the worm.” We see that “ate” is the root of the tree, and “the fish” is marked as the verb’s (nominal) subject, while “the worm” is marked as the verb’s (direct) object.

Head Finding Many methods for dependency parsing first perform constituency parsing. The dependency structure can then be determined through a process of *head finding* (Jurafsky and Martin, 2009).

Each syntactic constituent is associated with a lexical *head*, which is the word in the phrase which is grammatically the most important (Pollard et al., 1994). For example, the head of a verb phrase is the verb, and the head of a noun phrase is the noun. Heads are passed up the tree as shown in Figure 4.3. The remaining words are called *dependents*. As shown in Figure 4.2, a dependency tree marks these head-dependent pairs (Jurafsky and Martin, 2009).

The most common method for identifying the heads in a parse tree involves walking the tree and using a rule-based approach to determine the head at each non-terminal node (Jurafsky and Martin, 2009).

4.1.2.3 HPSG Parsing

Context free grammars suffer from several limitations when used to parse phrase structure. Many of these limitations arise from a lack of *lexical information* (information to do with individual words). Lexical features include, for example, *agreement* (e.g. number and gender) and *subcategorisation* (the number and type of arguments a verb may take) (Jurafsky and Martin, 2009). For example, “eats” is a transitive verb which usually has one argument (“The

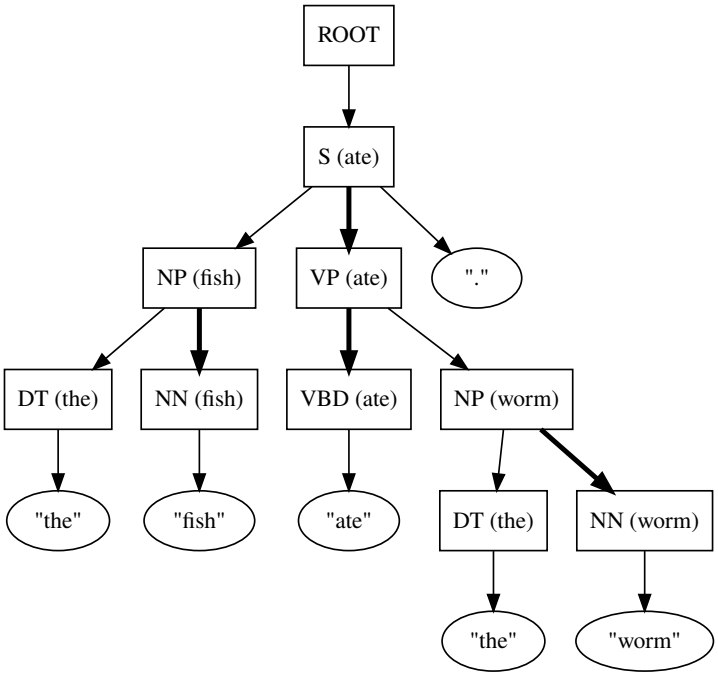


Figure 4.3: A constituency parse tree for the sentence "the fish ate the worm." including the head (shown in brackets) at each node. The heads are passed up the tree, as demonstrated by the bold arrows.

fish eats the worm"), while "gives" is a ditransitive verb which usually takes two arguments ("Jane gives Joan candy").

Lexicalised grammars rely more heavily on this kind of lexical information. A popular class of lexicalised grammars are *categorial grammars*, such as combinatory categorial grammar (CCG), which assigns tokens functional categories and defines rules for application, composition and type raising (Jurafsky and Martin, 2009). We will instead focus on a separate class of *constraint-based* grammar: head-driven phrase structure grammar.

Head-Driven Phrase Structure Grammar *Head-driven phrase structure grammar* (HPSG) (Pollard et al., 1994) is a highly-lexicalised, constraint-based grammar. HPSG has several important properties which strongly influence our own approach to representing natural language using answer set grammars:

1. **Based on typed feature structures.** HPSG replaces the atomic categories of a CFG with more complex data structures called *feature structures*. Each feature structure is a simple set of attribute, value pairs, where the value may be either atomic, or another feature structure (Muller and Sag, 2007). For example, a (simplified) feature structure of

type $word$ for the verb "eats" might look like:

$$\begin{bmatrix} word & \\ \text{PHON} & \langle \textbf{eats} \rangle \\ \text{HEAD} & \begin{bmatrix} verb & \\ \text{VFORM} & \textbf{fin} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \textbf{sg} \\ \text{PER} & \textbf{3} \end{bmatrix} \end{bmatrix} \\ \text{SUBCAT} & \langle NP[\textbf{nom}], NP[\textbf{acc}] \rangle \end{bmatrix}$$

Here, the HEAD feature structure encodes the syntactic category of the word: its a finite verb (VFORM **fin**) in its third-person (PER **3**) singular (NUM **sg**) form. The subcategorisation (SUBCAT) value tells us that eats is a transitive verb which must be assigned two arguments which are a nominative (subject) and an accusative (direct object) noun phrase.

2. **Highly lexicalised.** The grammar consists of a large and rich lexicon: each word is associated with a complex feature structure, such as the entry for "eats" shown above, which encodes phonological and syntactic information (such as the POS, agreement and subcategorisation), and often the word's semantics.

   Words have a different representation compared to phrases, which are made up of a HEAD-DTR (the head of the phrase) and NON-HEAD-DTRS (a list of the dependents) (Muller and Sag, 2007).

3. **Constraint-based.** HPSG replaces the generative rules of CFGs with a set of grammatical constraints which must be satisfied. For example a basic grammar rule might look like:

$$\begin{bmatrix} head\text{-}complement\text{-}structure & \\ \text{SUBCAT} & \boxed{1} \\ \text{HEAD-DTR} & \begin{bmatrix} sign & \\ \text{SUBCAT} & \boxed{1} \oplus \left\langle \boxed{2} \right\rangle \end{bmatrix} \\ \text{NON-HEAD-DTRS} & \left\langle \boxed{2} \right\rangle \end{bmatrix}$$

   where $\oplus$ represents concatenation, and values marked by the same boxed number must be identical (Muller and Sag, 2007). In short, this schema encodes "the using up of arguments". An possible instantiation could look like:

$$VP\,[\text{SUBCAT}\,\langle\rangle] \rightarrow NP[\textbf{nom}] \quad VP\,[\text{SUBCAT}\,\langle NP[\textbf{nom}]\rangle]$$

   Here, the *NP* is a non-head daughter and the *VP* is the head daughter. Since the category of the *NP* is the same as the last entry in the *VP*'s subcategorisation value, as per the grammar rule, we can form a phrase from these two constituents with a subcategorisation value that no longer requires the *NP* argument.

4. **Head-driven.** The constituent structure of HPSG follows the *head feature principle* (HFP):

$$\begin{bmatrix} headed\text{-}stucture & \\ \text{HEAD} & \boxed{1} \\ \text{HEAD-DTR}|\text{HEAD} & \boxed{1} \end{bmatrix}$$

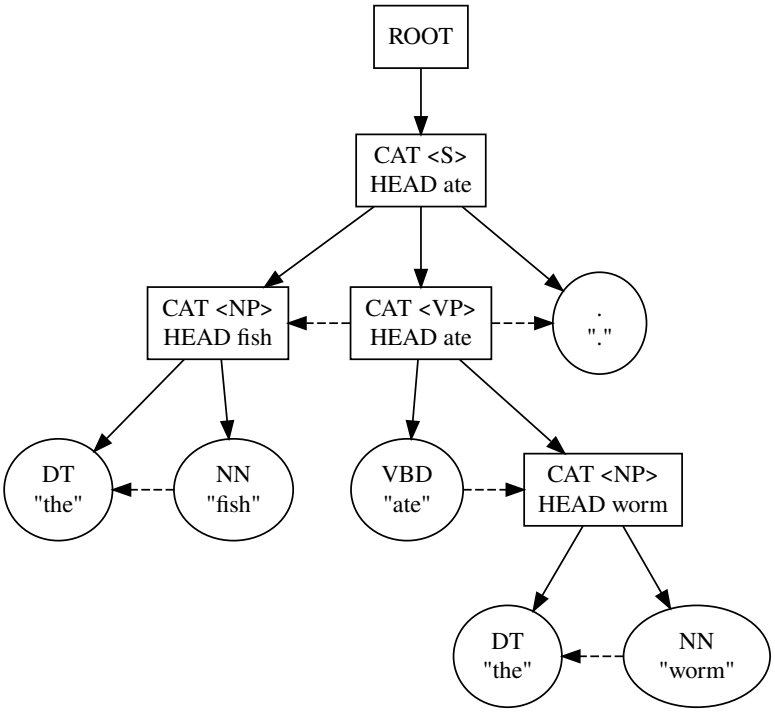


Figure 4.4: A joint span structure for the sentence "the fish ate the worm.". It encodes both the constituency (filled lines) and dependency (dashed lines) information.

> As per Pollard et al. (1994), this enforces that "the value of any headed phrase is structure-shared with the HEAD of the head daughter. The effect of HFP is to guarantee that headed phrases are really *projections* of their head daughter."

Note that the final point means that HPSG encodes *both* constituency (phrasal) *and* dependency structure (Zhou and Zhao, 2019). The former is useful for specifying and classifying sub-phrases within a text, and can easily be represented using a context free grammar, while the latter more transparently encodes predicate-argument structure and thus is much closer to representing the semantics of a text.

HPSG parsing is difficult in practice, as it is highly sensitive to the accuracy of the supertagger (which assigns lexical entries to each word): just one or two errors will often cause a parse failure. Thus most approaches use a CFG to approximate the HPSG (Matsuzaki et al., 2007), or use a heavily simplified representation (Zhou and Zhao, 2019) which ignores a lot of lexical information, and focuses mainly on the core of HPSG: the heads (we take the latter approach). An example of such a simplified structure, which focuses solely on the lexical head of each constituent, is shown in Figure 4.4.

4.1.3 Semantic Parsing

Semantic parsing is the process of assigning meaning representations (traditionally in the form of first-order logic formulae) to linguistic inputs (Jurafsky and Martin, 2009).

4.1.3.1 Semantic Representation

The first challenge of semantic parsing is finding a representation that reflects the meanings of texts in a transparent way.

Semantic Content As noted by Abend and Rappoport (2017), good semantic representations must be able to encode a large variety of semantic content. This content includes, but is not necessarily limited to:

1. **Events** (sometimes called frames, propositions or scenes), which include a predicate (main relation, frame-evoking element), arguments (participants, core elements) and secondary relations (modifiers, non-core elements) (Abend and Rappoport, 2017).

2. **Predicates**, which are the main determinants of what events are about. Most predicates are verbal, but some schemes also have nominal and adjectival predicates, e.g. by defining relations based on the noun "president". Several verbs may be assigned to the same predicate; for example, FrameNet assigns the same *ingestion* predicate to both of the verbs "eat" and "consume" (Ruppenhofer et al., 2006).

3. **Arguments**, which are usually distinguished between core and non-core. Core arguments are conceptually necessary components of an event, while non-core arguments introduce additional relations such as "time, place, manner, means and degree" (Ruppenhofer et al., 2006).

4. **Semantic roles**, which are categories of arguments. Some approaches use different semantic roles for different predicates. For example, FrameNet uses the semantic roles *ingestor* and *ingestible* for the *ingestion* predicate (Ruppenhofer et al., 2006). Other approaches define a closed set of abstract semantic roles, such as *agent* and *patient* (Kipper-Schuler, 2005).

5. **Discourse relations**, which mark the relations between events. Some examples include *succession* ("after"), *reason* ("since"), *juxtaposition* ("while") and *conjunction* ("and") (Prasad et al., 2008). Determining discourse relations is not always straightforward; for example, "since" may be used in either a temporal ("I haven't eaten since yesterday") or a causal ("I'm hungry since I haven't eaten") sense (or both). Proper understanding of these relations is vital for correct temporal, causal and spatial reasoning.

6. **Logical structure**, which handles issues such as quantification and negation (Abend and Rappoport, 2017). For example, a sentence such as "fish eat" implies universal quantification, $\forall f.fish(f) \rightarrow eats(f)$ while "the fish eats" implies existential quantification, $\exists f.fish(f) \wedge eats(f)$.

Event-Based Semantics Most modern schemes for semantic representation of text are based on *event semantics*. These schemes are distinguished by the notion that predicates take an implicit variable over events as an argument (Lasersohn, 2012). For example "the fish eats the worm" might be represented as

$$\exists e.eats(e, fish, worm) \tag{4.1}$$

instead of the more traditional representation

$$eats(fish, worm)$$

The formulation presented in (4.1) is usually referred to as a *Davidsonian* representation, after Davidson (1967). The motivation for representing sentences in this way is that modifiers may be treated simply as additional conjuncts: e.g. "the fish eats the worm at midnight" and "the fish eats the worm off the seabed" may be expressed as

$$\exists e.eats(e, fish, worm) \wedge at(e, midnight) \tag{4.2}$$

and

$$\exists e.eats(e, fish, worm) \wedge off(e, seabed) \tag{4.3}$$

respectively. This representation has multiple benefits: firstly, it avoids the ambiguity that arises from attempting to express (4.2) and (4.3) as $eats(e, fish, worm, midnight)$ and $eats(e, fish, worm, seabed)$ respectively — when clearly $midnight$ and $seabed$ are very different kinds of participants; secondly, it handles an arbitrary number of adjuncts in a straightforward manner; and finally, it naturally captures that (4.2) and (4.3) both entail (4.1).

The *neo-Davidsonian* approach (Parsons, 1990) further extends this idea by expressing the core arguments of the verb in a similar way: "the fish eats the worm" becomes

$$\exists e.eats(e) \wedge agent(e, fish) \wedge patient(e, worm) \tag{4.4}$$

There are two things to note here. The first is that the sentence "the fish eats" is now represented as $\exists e.eats(e) \wedge agent(e, fish)$ instead of $\exists ex.eats(e, fish, x)$. The latter implies that there is something that is eaten, whilst the former does not. The implication happens to be valid here, but this is not always the case. The second thing to note is that the predicates $agent$ and $patient$ in (4.4) express *thematic roles*, abstracting away from grammatical roles such as the subject or object of the sentence. Thus the passive construction "the worm is eaten by the fish" should share the representation of (4.4) — as we will see, this is a particularly useful quality of the approach.

Abstract Meaning Representation In recent years, many concrete schemes for semantic representation have been proposed for use in broad-coverage semantic parsers. One of the most widely adopted of these schemes is the Abstract Meaning Representation (AMR) (Banarescu et al., 2013), which implements a neo-Davidsonian semantics.

AMRs are directed acyclic rooted graphs, specifying the main events of a sentence using PropBank frames, and their arguments. Core arguments are assigned values 0 through 5, corresponding to agents, patients, instruments, starting points, ending points and modifiers respectively . Additionally, a number of general semantic relations are provided such as *age*, *beneficiary*, *cause*, *compared to*, *degree*, *example*, etc.

Example 4.2. (Abstract meaning representation) The AMR for "The boy wants to go." is shown in Figure 4.5.

Notably, AMR does not depend on syntactic structure, and in fact many approaches to generating AMRs do not make use of any syntactic features. The authors note that "he described her as a genius", "his description of her: genius" and "she was a genius, according to his description" should all be assigned the same AMR.

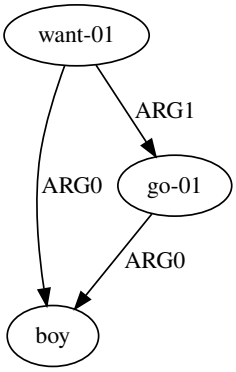


Figure 4.5: AMR for "The boy wants to go."

4.1.3.2 Lexical Semantics

Any compositional approach to determining the semantics of text must first address the issue of *lexical semantics*: how to represent and determine the semantics of individual words.

WordNet By far the most widely known and used resource for lexical information is *WordNet* (Miller, 1995). WordNet is a large lexical database of English. Its key contribution is to group English nouns, verbs, adjectives and adverbs into *synsets*, each of which expresses a distinct concept. It is important to note that the relationship between words and synsets is a many-to-many one: clearly one synset may contain multiple words, but the reverse is also true: one word can have *multiple senses* — this property is called *polysemy*. For example, the single synset `comfort.v.01` ("give moral or emotional strength to") consists of the lemmas *comfort*, *soothe*, *console* and *solace*, but the lemma *comfort* can also be assigned the sense `comfort.v.02` ("lessen pain or discomfort; alleviate").

In addition to assigning synsets, WordNet describes a small number of "conceptual relations" between synsets. These include:

1. **Hypernymy and hyponymy.** All verbs and nouns in WordNet are arranged in a hierarchy based on generality. A hyponym is a word whose semantic field is included in that of another word (i.e. it is more specific), whilst a hypernym is the reverse (more general). As an example, a hyponym of *eat* (consume food) might be *devour* (eat greedily) whilst the verb *consume* would be an example of a hypernym.

2. **Antonymy.** Adjectives are organised very differently to verbs and nouns. Instead of a hierarchical relationship, they are organised solely based on antonymy. Similar adjectives can be determined only by finding antonyms of antonyms, but we note that this process can result in many spurious matches. For example, using this method, the adjective *tasty* is deemed to be similar to *acid-tasting*!

3. **Entailment.** For verbs, WordNet also defines an entailment (implication) relationship: for example the verb *buy* entails *pay*.

4. **Meronymy.** For nouns, WordNet also defines a meronymy (is part of) relationship: for example *handle* is a meronym of *suitcase*.

5. **Cross-POS relations.** Finally, WordNet defines several cross-POS relations, mainly consisting of morphosemantic links that hold among semantically similar words sharing a stem with the same meaning (for example between the noun *eater* and the verb *eat*), although we find these are rarely useful since the relationship between these words is not usually specified.

Word Sense Disambiguation The issue of polysemy (one word being able to take multiple senses) gives rise to the problem of *word sense disambiguation* (WSD): how to determine a word's sense (usually WordNet synset) given its context (Jurafsky and Martin, 2009). There are various approaches to WSD, but current state of the art approaches use machine learning techniques to fine-tune pre-trained language models. WSD is, however, a difficult problem for machine learning approaches, particularly as it is extremely difficult to find enough labelled training data to cover each of WordNet's 117 000 synsets sufficiently. Indeed, even the current state of the art models, which use tricks such as sense compression to partly alleviate this issue (Vial et al., 2019), achieve an accuracy of 77.8% on SensEval 3 dataset. Meanwhile, simple heuristics alone, such as choosing the most common sense, achieve 66% accuracy on the same dataset.

4.1.3.3 Compositional Semantics

Compositional approaches to semantics are approaches which conform to the *principle of compositionality*, which states that the meaning of a phrase can be derived through some combination of the meanings of its sub-phrases (and thus eventually its individual words). This implies a close relationship between semantics and syntactic structure.

Semantic Parsing with CCG A common approach to compositional semantic parsing is to use CCGs (Bos, 2008; Chabierski et al., 2017; Reddy, 2017). In CCG, each word is assigned a category, which defines the expected types and positions of its inputs. For example, the determiner "the" might be assigned the category NP/N, meaning that when it precedes a noun, it forms a noun phrase. A set of combinatory rules can then be used to parse a sentence. For example, forward application ($>$) can be used to combine two constituents of type X/Y and Y to produce a single constituent of type X. An example of a full derivation is provided below:

$$\dfrac{\dfrac{Fish}{NP} \quad \dfrac{\dfrac{eat}{(S\backslash NP)/NP} \quad \dfrac{worms}{NP}}{S\backslash NP}\,>}{S}\,<$$

The main benefit of using CCG for semantic parsing is that it provides a transparent and well-studied syntax-semantics interface: it is possible to obtain the semantic properties of a phrase from its syntactic properties. To demonstrate this, let us return to our example:

$$\dfrac{\dfrac{Fish}{\lambda x.fish(x)} \quad \dfrac{\dfrac{eat}{\lambda fge.\exists xy.eat(e,x,y) \wedge f(y) \wedge g(x)} \quad \dfrac{worms}{\lambda x.worms(x)}}{\lambda ge.\exists xy.eat(e,x,y) \wedge worms(y) \wedge g(x)}\,>}{\lambda e.\exists xy.eat(e,x,y) \wedge worms(y) \wedge fish(x)}\,<$$

We note that lexical semantics, in terms of lambda calculus expressions, may be determined from a word's category. For example, a word with an atomic category such as fish (*NP*), is assigned a function with a single argument, $\lambda x.fish(x)$.

Combinatory rules are treated similarly: for example, the forward application rule corresponds to functional application in lambda calculus (Reddy, 2017):

$$X/Y : f \quad Y : g \quad \Longrightarrow_{>} \quad X : f(g)$$

In our example, *eat* and *worms* are combined as shown to produce

$$\underline{\lambda f g e.}\,(\exists xy.eat(e,x,y) \wedge f(y) \wedge g(x))\,\underline{(\lambda x.worms(x))}$$
$$\rightarrow_{\beta}\ \lambda ge.\exists xy.eat(e,x,y) \wedge worms(y) \wedge g(x)$$

The main drawback of using CCG for semantic parsing is the difficulty of accurate supertagging (the automated assignment of lexical categories), and its inability to cope with supertagging errors and ungrammaticality, which may make it impossible to find a valid derivation for a sentence (Reddy, 2017).

Semantic Parsing with HPSG An alternative grammar-based approach to semantic parsing is to use HPSG. HPSG has an integrated view of grammar, which expresses phonology, syntax and semantics in the same structure: feature structures are simply extended to encode this additional information (Muller and Sag, 2007). As before, this means that we rely heavily on the lexicon. Returning to our lexical entry for eats, we might encode the semantic content (CONT) as follows:

$$\begin{bmatrix} word & & \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} verb & \\ \text{VFORM} & \textbf{fin} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \textbf{sg} \\ \text{PER} & \textbf{3} \end{bmatrix} \end{bmatrix} \\ \text{SUBCAT} & \langle NP[\textbf{nom}]\boxed{1}, NP[\textbf{acc}]\boxed{2} \rangle \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} eat & \\ \text{AGENT} & \boxed{1} \\ \text{PATIENT} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

expressing that the nominative (subject) argument is the AGENT (eater) and the accusative (direct object) argument is the PATIENT (eaten) of the verb.

Compositionality of semantics is encoded by extending the grammatical constraints in a similar fashion. There are two key principles that guide HPSG's treatment of semantics (Sag and Wasow, 1999), both of which heavily influence our own treatment of semantics. Informally, these are:

1. **Semantic inheritance.** In headed phrases, the semantics are based on the head daughter's semantics.
2. **Semantic compositionality.** The context (entities and situation) described by a phrase corresponds to the sum of all the contexts described by *all* of its daughters.

Semantic Parsing from Dependency Structures Finally, let us discuss the problem of determining semantics from dependency structures alone. The closeness between dependency structures and semantics is well-known and has motivated the use of dependency parses as a substitute for full semantic parses in a variety of upstream tasks, such as co-reference resolution and question answering. However it is only recently that there has been any work on developing a semantics interface for dependency syntax. Reddy et al. (2016) propose a semantic parsing approach that uses lambda calculus to derive logical forms from dependency trees. The approach differs from CCG-based semantic parsers in two main ways: firstly, all constituents have the same type; and secondly, there is no distinction between core and non-core arguments. It consists of three main stages:

1. **Binarisation.** The dependency structure is mapped to a binary structure which indicates the order of semantic composition. For example, the sentence "the fish ate the worm" would be assigned the structure "(nsubj (dobj ate (det worm the)) (det fish the))".

2. **Substitution.** Each node in the binary structure is substituted with a lambda expression encoding its semantics. For example, the word "worm" would be assigned the expression $\lambda x.worm(x_a)$ and "ate" would be assigned $\lambda x.ate(x_e)$. The subscripts $\cdot_a$ and $\cdot_a$ denote the type **Ind** and **Event** respectively, whereas x denotes a paired variable (x_a, x_e). To simplify the type system, every constituent is assigned a type of **Ind** $\times$ **Event** $\rightarrow$ **Bool**. Expressions for dependency labels tend to take one of four forms:

   (a) **Copy.** $\lambda fgx.\exists y.f(x) \wedge g(y) \wedge rel(x, y)$. This kind of expression assigns a relation between two child nodes. For example, the dobj label in "(dobj ate (det worm the))" will result in a copy expression of the form $\lambda fgx.\exists y.f(x) \wedge g(y) \wedge arg2(x_e, y_a)$.

   (b) **Invert.** $\lambda fgx.\exists y.f(x) \wedge g(y) \wedge rel^i(y, x)$. This expression is used when the standard dependency direction is reversed — e.g. for the amod label in "(amod horse running)", where the event *follows* the individual.

   (c) **Merge.** $\lambda fgx.f(x) \wedge g(x)$. This expression is used to merge two sub-expressions without introducing any relations — e.g. for the amod label in "(amod horse beautiful)", where there is *no* event.

   (d) **Head.** $\lambda fgx.f(x)$. This expression simply returns the semantics of the parent expression. For example, it would be used to replace a punctuation label, "punct".

3. **Composition.** Finally, the logical form is computed by beta reduction. For example "(dobj ate (det worm the))" would result in $\lambda x.\exists z.ate(x_e) \wedge worm(z_a) \wedge arg2(x_e, z_a)$. This process continues up the tree, resulting in a final logical form of $\lambda x.\exists yz.ate(x_e) \wedge fish(y_a) \wedge worm(z_a) \wedge arg1(x_e, y_a) \wedge arg2(x_e, z_a)$ for the sentence "The fish ate the worm.".

Such an approach benefits from the pace of improvement in dependency parsing, as well as the simplicity of having nodes of a single type. In particular, the approach proposed by Reddy et al. (2016) was found to be extremely robust, with an 80% reduction in parser errors on the WebQuestions dataset, compared to a CCG-based approach.

4.2 Answer Set Programming

Answer set programming (ASP) is a form of declarative programming, primarily oriented towards difficult (NP-hard) search problems (Lifschitz, 2008). ASP programs consist of Prolog-like rules, but their semantics are based on the stable model semantics of logic programming (Gelfond and Lifschitz, 1988).

4.2.1 Answer Set Programs

4.2.1.1 Syntax of Answer Set Programs

The syntax of ASP is very similar to that of Prolog, with a few small modifications. Specifically, an ASP program consists of a set of rules:

Definition 4.2. (ASP rule) An ASP *rule* r has the form (Gebser et al., 2012):

$$H \leftarrow B_1, \ldots, B_m, \text{not } B_{m+1}, \ldots, \text{not } B_n.$$

We call

$$head(r) = H$$

the *head* of the rule. The head may be empty ($\bot$), an *atom* a, or an *aggregate* of the form $lb\{l_1, \ldots, l_k\}ub$, where each l_i is a *literal* (an atom, or the negation by failure of an atom), and lb and ub are integers[1]. If the head is empty, the rule is called a *constraint*. If the head is an aggregate, the rule is called a *choice rule*.

We call

$$body(r) = \{B_1, \ldots, B_m, \text{not } B_{m+1}, \ldots, \text{not } B_n\}$$

the *body* of the rule. Each body component B_i is either an atom or an aggregate. Also note that the set of body components may be empty; in this case r is called a *fact*.

4.2.1.2 The Stable Model Semantics

We now provide the definition of a *stable model* (or *answer set*), which provides a semantics for *grounded* ASP programs (ASP programs without variables) (Gelfond and Lifschitz, 1988). The first step to determining the stable models or answer sets of a program with variables is to eliminate the variables through a process of *relevant grounding*. This is a simple (although possibly infinite) process which finds all possibly useful groundings of the rules in a program.

In order to define the stable models for a *ground* program, we first explain the concept of a (minimal) Herbrand model (again, using definitions based on those in (Gebser et al., 2012)):

Definition 4.3. (Herbrand base) The *Herbrand base* $\mathcal{A}$ of a program P is the set of all ground atoms which can be made from predicates, functions and constants appearing in P.

[1] In reality, the head may be a disjunction over atoms, and aggregates may be more complex than what we present here. We omit these other possibilities for brevity.

Definition 4.4. (Herbrand interpretation) A *Herbrand interpretation* X of a program P assigns a truth value to every ground atom in the Herbrand base of P. Usually it is expressed as a set $X \subseteq \mathcal{A}$ of atoms specified to be true.

Definition 4.5. (Satisfaction) A rule r is *satisfied* by a Herbrand interpretation X (written $X \vDash r$) if $X \vDash head(r)$ or $X \nvDash body(r)$. The $\vDash$ relation is defined as follows:

1. $X \vDash a$ if $a \in X$.
2. $X \vDash$ not B if $X \nvDash B$ (*negation as failure*).
3. $X \vDash lb\,\{l_1, \ldots, l_k\}\,ub$ if $lb \leq |\{l_i | 1 \leq i \leq k, X \vDash l_i\}| \leq ub$.
4. $X \vDash body(r)$ if $X \vDash l_i$ for all $l_i \in body(r)$.

Definition 4.6. (Herbrand model) An interpretation X is a *Herbrand model* of P if $X \vDash r$ for every rule $r \in P$. A Herbrand model M of a program P is *minimal* if there is no model M' of P such that $M' \subset M$.

Note that if P is a definite logic program (contains no negation as failure), then there exists a unique minimal model, called the *least Herbrand model*, $M\,(P)$.

A *stable model* of a program is then defined as a minimal model of its own reduct:

Definition 4.7. (Reduct) The *reduct* of a ground ASP program P with respect to an interpretation X, P^X is constructed from every rule $r \in P$ whose body is satisfied by X, by:

1. Removing any negation as failure of body conditions whose atoms are not in X.
2. Removing any remaining rules which still have negation as failure conditions.
3. Replacing the rule's head, if it is an aggregate, with individual atoms belonging to X.

Note that P^X is a definite logic program, and so has a least Herbrand model.

Definition 4.8. (Stable model) X is a *stable model* of P if $M(P^X) = X$ (Gelfond and Lifschitz, 1988).

We will use the terms stable model and answer set interchangeably. Answer sets define the semantics of ASP programs. Specifically, an answer set program P is said to be *satisfiable* if it has at least one stable model, and *unsatisfiable* otherwise. Since a single answer set program may have multiple answer sets, there are two different kinds of semantics for whether an atom is *entailed* by a given program:

Definition 4.9. (Brave and cautious entailment) A program P *bravely entails* an atom a (written $P \vDash_b a$) if a is true in at least one answer set of P. A program P *cautiously entails* an atom a (written $P \vDash_c a$) if a is true in every answer set of P.

Example 4.3. (Answer sets and entailment) Consider the program:

$$P = \left\{ \begin{array}{c} a \leftarrow \text{not } b. \\ b \leftarrow \text{not } a. \\ c. \end{array} \right\}$$

This program has two answer sets: $\{a, c\}$ and $\{b, c\}$. This can be verified by considering the reduct of P with respect to each answer set. For example,

$$M(P^{\{a,c\}}) = M(\left\{ \begin{array}{c} a. \\ c. \end{array} \right\}) = \{a, c\}$$

Since a is true in at least one answer set, $P \models_b a$, but since it is not true in every answer set $P \not\models_c a$. c is true in every answer set and so $P \models_b c$ and $P \models_c c$.

4.2.2 Answer Set Grammars

Law et al. (2019) introduce answer set grammars (ASGs), which are a class of context sensitive grammars. These grammars are formed from context-free production rules with ASP annotations. These ASP annotations are able to express context-sensitive constraints, by means of *annotated atoms* which refer to atoms in a specific child of a node in the parse tree. Formally, an ASG can be defined as follows:

Definition 4.10. (Answer set grammar) An *answer set grammar* (Law et al., 2019) G is a tuple (G_N, G_T, G_{PR}, G_S) where:

- G_N is a set of *non-terminal nodes*, corresponding to symbols that are replaced by terminal symbols according to the production rules.
- G_T is a set of *terminal nodes*, disjoint from G_N, corresponding to symbols from the grammar's alphabet.
- G_{PR} is a set of *annotated production rules* of the form $n_0 \rightarrow n_1 \ldots n_k \; P$ where n_0 is a non-terminal node, n_1 through n_k are either non-terminal or terminal nodes, and P is an annotated ASP program, with integer annotations in the range $[1, k]$.
- G_S is a distinguished non-terminal node called the *start node*.

To explain the concept of *annotated atoms*, consider the following example from (Law et al., 2019):

Example 4.4. (Answer set grammar) The context-sensitive language $\texttt{a}^n\texttt{b}^n\texttt{c}^n$ can be defined by the ASG:

```
start -> as bs cs {
  :- size(X)@1, not size(X)@2.
  :- size(X)@1, not size(X)@3.
}
as -> "a" as { size(X+1) :- size(X)@2. }
as ->        { size(0). }
bs -> "b" bs { size(X+1) :- size(X)@2. }
bs ->        { size(0). }
cs -> "c" cs { size(X+1) :- size(X)@2. }
cs ->        { size(0). }
```

In this example, each of the non-terminal symbols `as`, `bs` and `cs` can be produced by two rules. The first rule produces each non-terminal symbol from an empty set of symbols, producing a node with just the fact `size(0)`. The second rule is more complex. `size(X)@2` means that the size of the string represented by the second child is `X`. So this rule adds a terminal symbol "a", "b" or "c" to the front of the string represented by the second child, and increments its size by 1.

The `start` node's ASP program encodes two constraints, namely that the number of a's (`size` of the first child) must not be different to the number of b's (`size` of the second child) and that the number a's (`size` of the first child) must not be different to the number of c's (`size` of the third child). A string is accepted by the grammar as long as it produces a parse tree that satisfies these two constraints. As such, this grammar accepts strings of the form $\texttt{a}^n\texttt{b}^n\texttt{c}^n$. A graphical representation of the accepted string "abc" is shown in Figure 4.6.

In general, an ASG accepts a string, if that string produces a parse tree with respect to the grammar, whose annotated ASP programs are satisfiable. Formally:

Definition 4.11. (Acceptance) A string of terminal nodes s is *accepted* by an answer set grammar (Law et al., 2019) G if there exists a parse tree T of G for s such that $G\,[T]$ is satisfiable, where

- $G[T]$ is the program $\{rule(n)@trace(n) | n \in T\}$.
- $rule(n)$ is the annotated production rule of node n in the tree T.
- $trace(n)$ is the parse trace of node n in the tree T. The trace of the root is the empty list $[]$, the ith child of the root is $[i]$, the jth child of the ith child of the root is $[i, j]$ and so on.
- For a production rule $r = n_0 \rightarrow n_1 \ldots n_k\ P$ and trace t, $r@t$ is constructed by:
  - Replacing all annotated atoms $a@i$ in P with the atom $a@(t +\!\!+ [i])$, and
  - Replacing all unannotated atoms in P with the atom $a@t$.

Example 4.5. (Acceptance) Let G be the grammar shown in Example 4.4 and T be the parse tree shown in Figure 4.6. Then, by translation each non-terminal node in T according Defini-

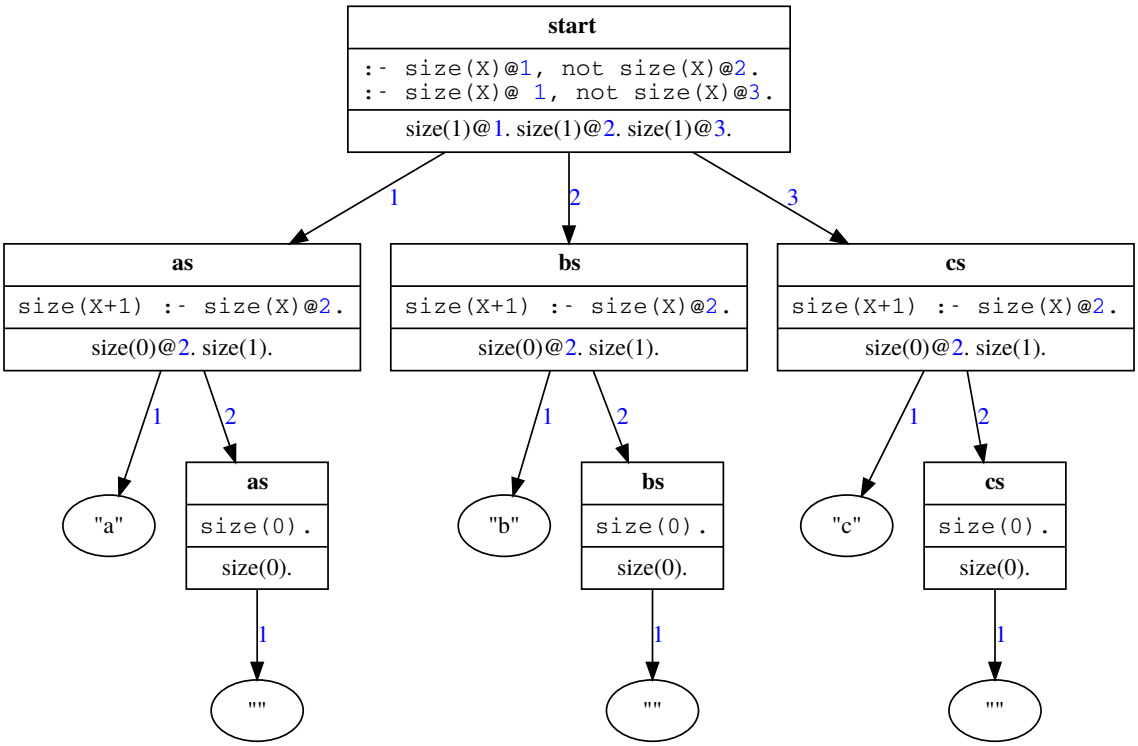


Figure 4.6: A graphical representation of the parse tree for the input “abc” using the grammar defined in Example 4.4. Non-terminal nodes are shown as tables; terminal nodes are shown as ellipses. For each non-terminal node, the first row gives the non-terminal symbol produced by the node, the second row gives the annotated ASP program at the node, and the third row gives the atoms which hold at the node.

tion 4.11 we have:

$$G[T] = \left\{ \begin{array}{c} \leftarrow size(X)@[1], \text{not } size(X)@[2]. \qquad \leftarrow size(X)@[1], \text{not } size(X)@[3]. \\ size(X+1)@[1] \leftarrow size(X)@[1,2]. \\ size(X+1)@[2] \leftarrow size(X)@[2,2]. \\ size(X+1)@[3] \leftarrow size(X)@[3,2]. \\ size(0)@[1,2]. \\ size(0)@[2,2]. \\ size(0)@[3,2]. \end{array} \right\}$$

Now we note that, by treating each annotated atom as a distinct atom, the program $G[T]$ has the single answer set:

$$AS(G[T]) = \left\{ \begin{array}{ccc} size(0)@[1,2] & size(0)@[2,2] & size(0)@[3,2] \\ size(1)@[1] & size(1)@[2] & size(1)@[3] \end{array} \right\}$$

Hence $G[T]$ is satisfiable, and so the parse tree T is *accepted* by G.

4.3 Inductive Logic Programming

Inductive logic programming (ILP) is a subfield of machine learning which uses first-order logic to represent hypotheses and data. In particular, it aims to find a hypothesis (a set of rules) which covers a set of positive examples and does not cover any negative examples, taking into account background knowledge (Muggleton, 1991).

Formally, given a language of possible hypotheses $\mathcal{L}_H$, a background theory B, a $covers$ relation specifying when an example is covered by a hypothesis (often taking into account the background knowledge), and a set of positive and negative examples $E = E^+ \cup E^-$, the goal is to find a hypothesis $H \in \mathcal{L}_H$ such that for all $e \in E^+$, $covers(B, H, e) = true$ and for all $e \in E^-$, $covers(B, H, e) = false$ (Raedt, 2017). This condition that *all* examples are correctly classified is sometimes weakened in order to handle noisy examples.

The most common setting for ILP is *learning from entailment*, where examples are ground facts, the $covers$ relationship is defined such that $covers(B, H, e) = true$ if and only if $B \cup H \models e$, and typically the hypotheses are definite clauses (Raedt, 2017). However, in this section we will focus on a more modern paradigm, called *learning from (partial) answer sets* (Law et al., 2014), which is an approach that integrates notions of brave and cautious semantics and finds hypotheses that are ASP programs.

4.3.1 Inductive Learning of Answer Set Programs

In this project, we use *ILASP*, a system developed by Law et al. (2015) which solves such learning from (partial) answer sets tasks.

4.3.1.1 Learning from Partial Answer Sets

In the learning from answer sets (ILP_{LAS}) paradigm (Law et al., 2014), examples are no longer ground facts, but instead *partial interpretations*.

Definition 4.12. (Partial interpretation) A *partial interpretation* e is a pair of sets of ground atoms (e^{inc}, e^{exc}) called the *inclusions* and *exclusions* respectively. A set of ground atoms X is said to *extend* the partial interpretation (e^{inc}, e^{exc}) if and only if $e^{inc} \subseteq X$ and $e^{exc} \cap I = \emptyset$.

The goal of a LAS task is to find an answer set program, that when combined with some background knowledge, has answer sets that extend all the positive examples, but no answer set which extends any negative example. Formally:

Definition 4.13. (Learning from answer sets) A learning from answer sets task T is a tuple (B, S_M, E^+, E^-) where B is an ASP program called the background knowledge, S_M is the hypothesis space defined by some language bias M and E^+ and E^- are positive and negative examples respectively.

A hypothesis H is said to be an inductive solution of T if and only if:

1. $H \subseteq S_M$.
2. For all $e \in E^+$, there exists an answer set A of $B \cup H$ such that A extends e.
3. For all $e \in E^-$, there does not exist any answer set A of $B \cup H$ such that A extends e.

Note that the ILP_{LAS} task can express both brave induction tasks (by use of a single positive example) and cautious induction tasks (using negative examples).

Also note that in order to help make the search process tractable, the search space for the inductive hypothesis is constrained to S_M. This is achieved by providing a language bias M specified by mode declarations. For example, a $modeh$ declaration declares the kinds of ground atoms that may appear in the head of a learned rule, and a $modeb$ declaration declares the kinds of ground atoms that may occur in the body of a learned rule.

Example 4.6. (Mode declarations) Given the mode declarations:

$$M = \left\{ \begin{array}{c} modeh(p) \\ modeb(q) \end{array} \right\}$$

the hypothesis space would be limited to:

$$S_M = \left\{ \begin{array}{ccc} & \leftarrow q. & \leftarrow \text{not } q. \\ p. & p \leftarrow q. & p \leftarrow \text{not } q. \end{array} \right\}$$

4.3.1.2 Learning from Context-Dependent Examples

In traditional ILP approaches, all examples must be explained by a single hypothesis and shared background knowledge. However, sometimes examples may be context dependent; i.e. they should have slightly different background knowledge. An extension to the ILP_{LAS} task, called *context-dependent learning from answer sets* $ILP^{context}_{LAS}$ (Law et al., 2016) allows the use of contexts to structure the background knowledge in this way. This increases the flexibility and often the performance of the learning task.

In context dependent learning from answer sets, examples become context-dependent partial interpretations (CDPIs):

Definition 4.14. (Context-dependent partial interpretation) A *context-dependent partial interpretation* (Law et al., 2016) is a pair $e = (e_{pi}, e_{ctx})$ where e_{pi} is a partial interpretation and e_{ctx} is an ASP program. An answer set X of $P \cup e_{ctx}$ to be an *accepting answer set* of e with respect to P if and only if X extends e_{pi}.

The $ILP^{context}_{LAS}$ learning task itself is defined in the same way as the ILP_{LAS}, except answer sets must now be *accepting answer sets* of each example e, instead of *extending* each example e as in Definition 4.13.

4.3.2 Answer Set Grammar Induction

Using the ILASP system, Law et al. (2019) demonstrate an approach for learning answer set grammars. In particular, they propose a framework that takes as an input an ASG G, a hypothesis space S_M, and two sets of strings E^+ and E^- called the positive and negative examples respectively, and learns an ASG G' (with the same context-free grammar as G) such that G' accepts every string in E^+ and none from E^-.

Informally, this approach is intended to learn the semantic constraints of a language (in the form of ASP annotations), given the syntactical form of the language (defined by the CFG of the *existing grammar* G) and a set of semantically valid and invalid examples (E^+ and E^- respectively). This is done by translating the ASG learning task into an equivalent $ILP^{context}_{LAS}$ task, as described in Definition 4.15.

Let us note that the representation of an ASG learning task is slightly different to the ILP frameworks we have seen so far. Firstly, its examples are strings instead of partial interpretations. Another important modification is that the mode declarations which form the hypothesis space S_M have an additional parameter called the *scope*. The scope specifies the production rules to which the mode declaration applies. For example, the mode declaration $\#modeb(p, [1, 2])$ is used to enforce that the atom p may appear in the body of rules learned in the two annotated ASP programs belonging to the first two production rules for an ASG. A limit is also applied to the depth of the parse tree in order to make the task tractable.

Definition 4.15. (ASG learning task) An ASG learning task (Law et al., 2019) $\langle G, S_M, E^+, E^- \rangle$ is equivalent to the $ILP^{context}_{LAS}$ task $\langle B, S^{LAS}_M, E^+_{LAS}, E^-_{LAS} \rangle$ where:

- Each production rule PR in G takes the form $n \rightarrow n_1 \ldots n_k$ P and has its own unique ID PR_{id}. The first production rule is assigned the ID 1, the second 2, and so on.

- Let $R_X\left(PR_{id}\right)$ be the rule formed from R by:
  - Replacing each annotated atom $a@[t_1,\ldots,t_n]$ with the atom $a@X +\!\!+ [t_1,\ldots,t_n]$ (and treating each annotated atom as a distinct atom, as before).
  - Adding the atom $node\_rule(PR_{id}, X)$ to the body of the rule (this is used to determine which production rule is used at each node in the parse tree).
- B is then defined to be the set of rules $R_X\left(PR_{id}\right)$ for each ASP rule $R \in P$ in each production rule PR in G.
- S_M^{LAS} is defined similarly to be the set $\{R_X\left(PR_{id}\right) | \langle PR_{id}, R\rangle \in S_M\}$. Note that here each PR_{id} defines the *scope* for each rule R in the hypothesis space.
- E^+_{LAS} contains one CDPI $\langle\langle\emptyset,\emptyset\rangle, C\rangle$ for each string $s \in E^+$. Given a set of possible parse trees $PT_1,\ldots,PT_m$ for s, then C ensures that at least one of these parse trees is accepted. Specifically, C is formed of the rules:
  - $1\,\{pt_1,\ldots,pt_m\}\,1$ (chooses one parse tree).
  - $node\_rule(rule(n)_{id}, trace(n)) \leftarrow pt_i$ for each $i \in [1..m]$ and for each node $n \in PT_i$ (ensures satisfiability of the chosen parse tree).
- E^-_{LAS} is a set of CDPIs $\langle\langle\emptyset,\emptyset\rangle, \{node\_rule(rule(n)_{id}, trace(n)) | n \in PT\}\rangle$ for each parse tree PT for each string in E^-, thus ensuring that no such parse tree is accepted.

Example 4.7. (ASG learning task meta-representation) Let G be the grammar shown in Example 4.4. Then by translating each production rule according to Definition 4.15 we get:

$$B = \left\{\begin{array}{c} \leftarrow size(X)@Y +\!\!+ [1], \text{not } size(X)@Y +\!\!+ [2], node\_rule(1, Y).\\ \leftarrow size(X)@Y +\!\!+ [1], \text{not } size(X)@Y +\!\!+ [3], node\_rule(1, Y).\\ size(X+1)@Y \leftarrow size(X)@Y +\!\!+ [2], node\_rule(2, Y).\\ size(0)@Y \leftarrow node\_rule(3, Y).\\ size(X+1)@Y \leftarrow size(X)@Y +\!\!+ [2], node\_rule(4, Y).\\ size(0)@Y \leftarrow node\_rule(5, Y).\\ size(X+1)@Y \leftarrow size(X)@Y +\!\!+ [2], node\_rule(6, Y).\\ size(0)@Y \leftarrow node\_rule(7, Y). \end{array}\right\}$$

Now suppose the string "abc" is provided as a positive example. "abc" has a single parse tree, which is the one shown in Figure 4.6. Hence it would produce the context:

$$C = \left\{\begin{array}{c} 1\{pt_1\}1.\\ node\_rule(1, []) \leftarrow pt_1.\\ node\_rule(2, [1]) \leftarrow pt_1.\\ node\_rule(3, [1, 2]) \leftarrow pt_1.\\ node\_rule(4, [2]) \leftarrow pt_1.\\ node\_rule(5, [2, 2]) \leftarrow pt_1.\\ node\_rule(6, [3]) \leftarrow pt_1.\\ node\_rule(7, [3, 2]) \leftarrow pt_1. \end{array}\right\}$$

Note how $B \cup C$'s single answer set is identical to $G\,[T]$'s answer set from Example 4.5.

Chapter 5

Representing Natural Language

In this chapter, we discuss our approach to translating a set of natural language sentences into a base answer set grammar, which is capable of parsing and determining the semantics of any sentence with the same linguistic structures. We firstly describe and motivate the design of a formal graph-based representation for the semantics of natural language. We then present an automated approach to constructing these graphs using existing deep-learning-based NLP techniques to automatically generate base answer set grammars that are able to produce them.

Qualities of a Good Representation The primary focus of this chapter is to find an expressive formal representation for natural language texts that can be automatically generated. Bearing in mind that we intend to use the learning from answer sets framework to learn commonsense rules related to the texts, a good representation should have the following qualities:

1. **Structured.** Since our learning method is logic-based, it is necessary to define some predicate structure.
2. **Semantics-based.** Since we want our results to be explainable, we must ensure that our representation is adequately close to the text's semantics. Otherwise, explanations are likely to be both shallow and difficult to interpret.
3. **Has canonical forms.** Furthermore, it is desirable that sentences that have the same semantics have the same representation, even if their syntax differs. This concept is key to our logic-based learning approach: in order for a logical rule to generalise to similar examples, we must be able to specify the similar part of those examples, and as such they need to have the same logical representation.
4. **Unambiguous.** Words and phrases can be ambiguous: that is, they have different meanings according to their context. Our representation should not introduce any more ambiguities than those that exist in the original text. Ambiguities make learning much harder, as the learner will be required to resolve them in addition to learning the desired knowledge.
5. **Forms small hypothesis spaces.** A final challenge is that of making sure that the representation is well-suited to automated approaches for generating hypothesis spaces

for learning. In particular, we want to minimise the number of literals, variables and constants generally necessary to explain a solution. This prescribes that we use a small number of general predicates, minimise the need for variables to be shared between predicates, and minimise the number of constants in our representation.

We note that the direct outputs of the most common existing NLP processing techniques are not well-suited to logic-based learning approaches. Most state-of-the-art approaches, while generally able to capture some of the semantics of a text, are based on vector embeddings and thus fall at the first hurdle: they do not have any clear predicate structure. More traditional approaches, such as parsing, offer the structure we require, but their results tend to be much more distanced from the semantics of the text.

5.1 Knowledge Graphs

We propose a graph-based representation scheme, in which each text is represented by what we will call a *knowledge graph*. This graphical structure is influenced both by previous work on the WSC (Schüller, 2014; Sharma, 2019) and popular representations for broad-coverage semantic parsing (Banarescu et al., 2013), however we make significant changes, both to improve the ease of its automatic construction and ease of use for inductive logic programming. These changes largely build upon previous work done by Reddy et al. (2016) and Chabierski et al. (2017) respectively.

Predicate Structure We propose a predicate structure which describes an acyclic, directed graph representation for a sentence.

Example 5.1. (Predicate structure) The sentence "Jim yelled at Kevin because he was so upset." would be mapped to the following logical structure

$$
\begin{gathered}
event(e1, yell, n1) \\
\wedge event(e2, upset, n3) \\
\wedge nominal(n1, person) \\
\wedge nominal(n2, person) \\
\wedge nominal(n3, he) \\
\wedge modifier(m1, at, e1, n2) \\
\wedge modifier(m2, because, e1, e2) \\
\wedge modifier(m3, so, e2)
\end{gathered}
$$

We call the first argument of each literal the *identifier* of that literal, the second its *lemma* and the rest its *core arguments.*

This approach closely resembles the classical Davidsonian approach to event semantics, but similarly to Chabierski et al. (2017), we significantly reduce the number of predicates in our representation by making the lemma an argument of each literal, rather than the predicate

name, which instead corresponds to one of a small number of semantic categories (i.e. we write $nominal(n1, person)$ rather than $person(n1)$). Our representation differs from the Chabierski et al. (2017) approach in both the number and kinds of categories. We use just three predicates: *events* capture the main verbal or adjectival meaning of a phrase, *nominals* are used for core arguments (i.e. generally corresponding to noun phrases), and *modifiers* are used both for non-core arguments (such as prepositional modifiers) and for linking events together (i.e. discourse connectives).

Each **event** has up to four core arguments. In order, these are the subject, direct object, indirect object, and what we refer to as the "control argument". The control argument is used to handle control verbs such as *tried* in "Paul *tried* to call George," which we represent as[1]

$$event(e1, try, paul, e2) \wedge event(e2, call, paul, george)$$

In our representation, the first three core arguments must be nominals, whilst the last, where present, must be an event.

Each **modifier** has either one or two core arguments, which may be of any type. The first is the event/nominal/modifier being modified, and the second, where present, is the object of the modifier.

Nominals make up the most straightforward category as they do not have core arguments, just an identifier and a lemma.

Reasoning behind our Predicate Structure This design is significantly influenced by the need to make the hypotheses generated by a learning approach as simple as possible. The implementation of a Davidsonian semantics rather than a neo-Davidsonian one means that in many cases, we generate rules with fewer literals and variables. For example the knowledge that "the person who is upset is the person who yells" could be expressed simply as

$$same(N1, N2) \leftarrow event(\_, yell, N1), event(\_, upset, N2)$$

rather than

$$same(N1, N2) \leftarrow event(E1, yell), event(E2, upset), agent(E1, N1), agent(E2, N2).$$

Meanwhile, the advantages of making the lemma an argument rather than a predicate name are twofold. Firstly, this representation is more expressive since the lemma may be replaced by a variable. This allows us to express, for example, knowledge such as "a person hires somebody to *do something* for them"

$$same(N1, N2) \leftarrow event(\_, hire, N1, \_, E1), event(E1, \boxed{\_}, \_), modifier(\_, for, E1, N2).$$

which is only possible as the boxed anonymous variable allows us to refer to *any* event. Secondly, this change makes the definition of the hypothesis space more straight forward by adding an additional layer of abstraction through use of constants. If, for example, we wish to allow some adjective to be modified by *either* "less" *or* "more", rather than requiring two mode declarations $less(var(m), var(e))$ and $more(var(m), var(e))$, we can instead define a single mode declaration with an appropriate constant: $modifier(var(m), const(c), var(e))$.

[1]Note that in reality, it is necessary to add placeholder arguments in any unused slots so that core arguments of the same kind are always in the position and can be matched, however we omit them here for brevity and clarity. This means that the try event in "Paul tried to call George", for example, would actually be represented as $event(e1, try, paul, null, null, e2)$.

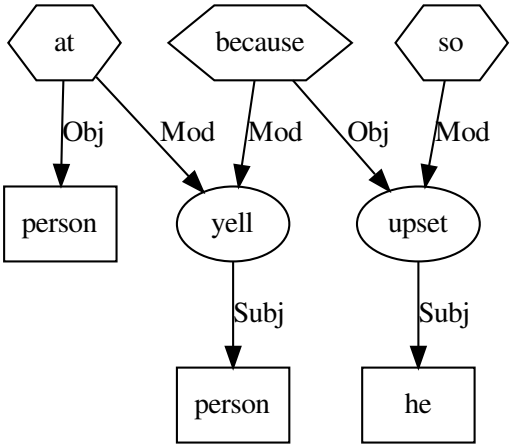


Figure 5.1: Knowledge graph corresponding to Example 5.1. We represent events with ellipses, nominals with rectangles, and modifiers with hexagons.

Knowledge Graphs Knowledge graphs are simply graphical representations of the predicate structure we have described so far.

Example 5.2. (Knowledge graph) The predicate structure of Example 5.1 corresponds to the knowledge graph shown in Figure 5.1.

Each *node* in the knowledge graph corresponds to an *identifier* in the logical representation, and each outgoing *edge* from a node corresponds to a *core argument* of the literal with that node's identifier. We represent events with ellipses, nominals with rectangles, and modifiers with hexagons.

5.2 Answer Set Grammars for Natural Language

In this project, we make use of answer set grammars to represent natural language. We propose that these answer set grammars are used to simultaneously perform a syntactic and semantic parse of a text, generating both a phrase-structure parse tree and knowledge graph structure respectively.

The motivation for this approach is threefold:

1. **Syntax-mediated semantics.** Firstly, any approach which uses a grammar-based approach to determine the semantics of a text benefits from intrinsic structural knowledge of the text. Previous work (Reddy, 2017) has demonstrated that this structural knowledge is useful for accurately determining the semantics of the text, due to the close relationship between syntax and semantics. Additionally, it forces the semantics to be encoded in a compositional way which can be easily understood by the user.

2. **ASP rules and constraints.** Secondly, ASG-based approaches additionally benefit from being able to use ASP rules to express not only semantic composition, but also constraints over both the syntax and semantics of texts accepted by the grammar. Importantly, since the ASG itself encodes the compositionality of the semantics of the text, these rules and constraints can be applied not only to the text as a whole, but also at the phrase-level. In contrast to most existing constraint-based grammars used in language processing, the computational foundation of ASGs, answer set programming, is widely-used and well-supported by a wide range of sophisticated tooling. In particular, inductive logic programming systems such as ILASP can be used to learn semantic rules and constraints directly within the grammar, which is an idea that we explore in detail in this report.

3. **Reusability.** Finally, and most importantly, we see the main benefit of ASGs in natural language processing as *domain-specific semantic parsers*. That is, since an ASG encodes how to build up the semantics from individual words, we can reuse it to determine the semantics of different, previously unseen sentences, as long as the *linguistic structures* of those sentences were considered when building the base ASG. This has an additional benefit when considering *learning* with answer set grammars. If a system such as ILASP is used to extend an ASG with automatically learned knowledge or constraints, these newly-learned rules or constraints can then be applied directly to any parseable sentence, without the need to first run complex machine learning models to generate a semantic parse.

To demonstrate our ASG-based formalism for natural language, and support the claim that it can be used to build domain-specific semantic parsers, let us consider a very simple example motivated by the bAbI tasks dataset.

Example 5.3. (ASG for natural language) A very simple answer set grammar capable of performing a semantic parse of the pattern "{person} {went/travelled/moved} to the {location}." could take the form

```
% Grammar
start -> np vp "." {
  event(ID, Lemma, Arg0) :- event(ID, Lemma)@2, nominal(Arg0, _)@1.
  nominal(ID, Lemma) :- nominal(ID, Lemma)@1.
  modifier(ID, Lemma, Arg0, Arg1) :- modifier(ID, Lemma, Arg0, Arg1)@2.
  nominal(ID, Lemma) :- nominal(ID, Lemma)@2.
}

np -> nnp {
  nominal(ID, Lemma) :- nominal(ID, Lemma)@1.
}

vp -> vbd pp {
  event(ID, Lemma) :- event(ID, Lemma)@1.
  modifier(ID, Lemma, Arg0, Arg1) :- modifier(ID, Lemma, Arg1)@2,
                                     event(Arg0, _)@1.
  nominal(ID, Lemma) :- nominal(ID, Lemma)@2.
}
```

```
pp -> to np {
  modifier(ID, Lemma, Arg1) :- modifier(ID, Lemma)@1,
                               nominal(Arg1, _)@2.
  nominal(ID, Lemma) :- nominal(ID, Lemma)@2.
}

np -> dt nn {
  nominal(ID, Lemma) :- nominal(ID, Lemma)@2.
}

% Lexicon
nnp -> "Mary"       { nominal(mary, person).     }
nnp -> "John"       { nominal(john, person).     }
nnp -> "Daniel"     { nominal(daniel, person).   }

vbd -> "went"       { event(went, go).           }
vbd -> "travelled"  { event(travelled, go).      }
vbd -> "moved"      { event(moved, go).          }

to -> "to"          { modifier(to, to).          }
dt -> "the"         {                            }

nn -> "bedroom"     { nominal(bedroom, bedroom). }
nn -> "garden"      { nominal(garden, garden).   }
nn -> "kitchen"     { nominal(kitchen, kitchen). }
nn -> "office"      { nominal(office, office).   }
```

The **grammar rules** encode how to compose the semantics of the words in the sentence. To see this, let us consider the first production rule (`start -> np vp ‘‘.’’`). Here, the first ASP rule sets the agent of the event from the verb phrase (*goes*) to be the nominal from the noun phrase which precedes it (*the person* who goes). The remaining three rules simply pass up the semantics of the *person*, *location* and prepositional modifier (goes *to* the location), which are all finalised at lower nodes in the parse tree by other rules. Let us also consider the correspondences between our approach and the HPSG formalism: in every production rule, the literal at the head of the first ASP rule comes from the head daughter — this literal gives the actual meaning of each constituent, as per the *semantic inheritance principle,* and is the part which is modified by any dependents; the remaining ASP rules then encode the *semantic compositionality principle*, by combining all of the existing context from *all* of the constituent's children.

The **lexicon** encodes the semantics of each individual word. As we will see, our approach is not really limited by the lexicon, since lexical entries are extremely straightforward to generate automatically, so unseen vocabulary is not a significant issue for the approach.

We note that, as originally claimed, this grammar is capable of doing a semantic parse of any sentence of the form “{person} {went/travelled/moved} to the {location}.” For example, “Mary went to the bedroom.” produces the semantic representation

$$event(went, go, mary) \wedge modifier(to, to, went, bedroom) \\ \wedge nominal(mary, person) \wedge nominal(bedroom, bedroom)$$

at the root node. As we will demonstrate in this report, this approach can be extended to automatically handle much more difficult patterns with large vocabularies and many complex linguistic structures.

5.3 Automated Translation from Dependency Structures

We now detail an approach for generating knowledge graphs, using automatically-generated answer set grammars to encode semantic composition.

Qualities of a Good Translation Approach We initially identified four main requirements for the translation approach.

1. **Grammatical compatibility.** Since we intend to learn with ASGs, it is necessary to construct a base CFG with enough semantic annotations to allow relevant semantic constraints to be learned. Thus the ASG production rules need to encode both syntactic and semantic composition, and the latter must therefore be linked cleanly to the former.

2. **Domain-generality.** Winograd Schemas use a large vocabulary and are not constrained to any specific domain, thus our approach must be domain general.

3. **Robustness.** The WSC includes a very diverse set of linguistic constructions, some of which are complex. For example negation, conjunction, relative clauses, passives, raising and control are all constructions that occur frequently in the WSC. Furthermore, many of the examples that we find through automated approaches for knowledge hunting may actually be ungrammatical. It is important that our approach can handle all of these complex and ungrammatical texts gracefully.

4. **Accuracy.** It is vital that the automatically generated graphs are very often correct. Since we intend to use a logic-based learning approach, even small amounts of noise in the final results can make knowledge contradictory, causing our learning tasks to become unsatisfiable and hence hindering our ability to learn anything.

Dependency-Guided Semantic Composition We settled on an approach that uses both constituency and dependency structures, inspired by the HPSG formalism. The constituency structure is used mostly to determine syntactic composition and generate the base CFG, and so we will largely ignore it here. Meanwhile the dependency structure is largely used to determine how to compose the semantics, and generate the knowledge graph structure itself. This part of the approach is influenced by the previous work of Reddy et al. (2016), although our final representation is very different to theirs, and we use the ASG directly to encode the composition of semantics, rather than using intermediate forms based on lambda calculus.

The motivation behind using the dependency structure is twofold. Firstly, its close relatedness to the constituency structure (which we highlighted in Subsection 4.1.2.2) is a useful property with regards to the goal of grammatical compatibility, since the constituency structure is very easily expressed with a CFG. Secondly, there are many large dependency treebanks

which have led to the development of highly robust and accurate parsers. State-of-the-art dependency parsers achieve accuracies of over 96% for labelled dependencies, and over 97% accuracy for unlabelled ones. Traditionally, the accuracy of alternatives, such as CCG parsing, has somewhat lagged behind, presumably due to training data being less widely available. Previous work (Reddy, 2017) has shown that this translates to dependency-based approaches outperforming their CCG-based counterparts on the task of semantic parsing, particularly for complex and ungrammatical texts.

Our dependency-guided approach consists of four stages:

1. **Dependency parsing.** We use existing machine learning models for tokenisation, lemmatisation, POS-tagging, and constituency and dependency parsing. The exact choice of models is discussed in Section 7.1.

2. **Lexicon generation.** A lexicon is generated which substitutes each token for a literal which encodes the type and meaning of the token, as well as any other lexical features.

3. **Grammar generation.** A grammar is generated which substitutes each dependency structure for a production rule with ASP annotations which compose the semantics of its children.

4. **Composition.** The knowledge graph structure is generated by parsing the original text with the answer set grammar. At each node in the parse tree, the set of atoms that hold express the semantics of the phrase parented by the node. The atoms which hold at the root node express the semantics of the full text.

Lexicon The first stage of our approach generates a lexical entry for each word in the original text. Currently this is a simple process that uses the token's POS tag and lemma to generate a base literal of the correct type with no core arguments.

Example 5.4. (Lexical entry) The word "yelled" in the text "Jim yelled at Kevin because he was so upset ." produces the lexical entry

```
vbd -> "yelled_1" {
  event(yelled_1, yell, null, null, null, null).
}
```

There are several things to note here.

The first is that the entry has the *event* type, which is deduced based on its POS tag being VBD (verb, past tense). Each POS tag is associated with a type: verbs and adjectives create events, nouns and pronouns create nominals, and most other tags, including adverbs, prepositions, subordinating conjunctions and coordinating conjunctions create modifiers. Certain tags, such as most punctuation tags, are ignored entirely.

The second is that the event's identifier is `yelled_1`. This is formed of the token name for readability of the generated grammar, as well as the token's index in the original sentence to ensure that the same word does not receive the same identifier when used in several positions.

The third is that the second argument, `yell`, comes directly from the token's lemma. To handle proper nouns, where the lemma is unlikely to capture any useful semantic information, we instead assign an NER class of `person`, `organization` or `location`. Tense, person, number, mood, aspect etc. are currently all ignored. This allows us to match the meaning of words, even when they are present in different forms, which expands the amount of knowledge that might be applicable to a certain problem. If this additional lexical information were to be added to the lexicon, then it would be provided through additional arguments to ensure this property is maintained. However, we found that such information is almost never useful for the problems we choose to focus on.

Finally, we note that the event has four `null` core arguments. There are four arguments here since, as discussed earlier, an event literal may have up to four core arguments. For a modifier there would be two, and for a nominal, none. These arguments contain a placeholder value so that we can tell which ones have been assigned values. This property is useful thanks to a linguistic principle named the *theta criterion* (Barany, 2017), which states that each thematic role is assigned one and only one argument. Thus if a core argument has been assigned, we know that it is the only such argument and that it can be treated as final. Further subcategorisation information can easily be added by using a different value for core arguments that are never used, and/or additional literals to assert properties of certain arguments. For example, a non-control verb which is monotransitive and hence never has an indirect object or control argument may instead have an entry of the form `event(ate_2, eat, null, null, unused, unused)`. However in practice, we find that such information is neither useful nor easy to determine accurately.

A nice property of our simple lexicon is that it is very straightforward to generate new entries: minimally, just a POS tag and lemma is required. This enables us to handle unseen vocabulary without too much difficulty.

Grammar Rules In our approach, grammar rules encode both syntactic and semantic composition.

Example 5.5. (Grammar rule) The `nsubj` (nominal subject) dependency between the event "yelled" and the nominal "Jim" in the sentence "Jim yelled at Kevin because he was so upset ." produces the grammar rule

```
s -> np vp {
  event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)
    :- event(ID, Lemma, null, Arg1, Arg2, ArgCtl)@2,
       nominal_unused(Arg0, _)@1.
  nominal(ID, Lemma)
    :- nominal_unused(ID, Lemma)@1.
  ...
}
```

Here, the first rule sets the previously unassigned subject of the verb phrase to be the nominal produced by the noun phrase, while the second rule "uses up" the nominal argument. We mark literals unused until they are either used up as an argument or they reach the root node (which

may happen if the text is ungrammatical). This design enforces a second constraint imposed by the *theta criterion* (Barany, 2017): namely that each argument is assigned to one and only one thematic role.

The two ASP rules shown here are followed by rules to pass up any events, modifiers and nominals which have already been finalised at nodes lower in the parse tree. This ensures that the atoms which hold at the root node together form a conjunction that fully expresses the semantics of the whole text.

Lexicalised Grammar Rules

Example 5.6. (Lexicalised grammar rule) An alternative formulation for the grammar rule presented in Example 5.5, with a close resemblance to a rule of a lexicalised CFG, is

```
s -> np vp {
  event(ID, yell, Arg0, Arg1, Arg2, ArgCtl)
    :- event(ID, yell, null, Arg1, Arg2, ArgCtl)@2,
       nominal_unused(Arg0, person)@1.
  nominal(ID, Lemma)
    :- nominal_unused(ID, Lemma)@1.
  ...
}
```

where, in the first rule, we replace the variable `Lemma` with the constant `yell` and the anonymous variable with the constant `person`. The advantage of such an approach is that the grammar is less likely to accept ungrammatical sentences or produce several incorrect parse trees[2]. The second property is particularly useful since learning tasks are made considerably simpler by having only a single parse tree, as each parse tree corresponds to a different possible answer set in the learning task's meta-representation. Clearly, the main drawback of this lexicalised version is that the produced rule is a lot less general and so the produced ASG is a lot less likely to be able to parse unseen texts. In particular, adding entries to the lexicon would no longer be sufficient to handle unseen vocabulary: the grammar rules would also need to be updated to encode which event-argument pairs are valid.

Automatic Generation of Grammar Rules The rules shown above are generated automatically from the text's dependency structure. This is done in three stages: for each constituent we firstly *identify* the dependencies which are present, then *translate* each of these dependencies into a logical intermediate representation, and finally *combine* the dependencies to form the correct grammar rule.

At each constituent, we *identify* the head daughter and each dependent. The translation then depends on the type of the head daughter, the type of the dependent node, and the dependency label. The types of the head and dependent are checked against the label: for example, an

[2]In particular, note how lexicalised grammar rules automatically capture selectional restrictions — semantic constraints on arguments that account for the implausibility of sentences such as "colourless green ideas sleep furiously".

Head		Dependent: Nominal (NN, NNS, NNP, ...)	Dependent: Event (VBD, VBZ, JJ, ...)	Dependent: Modifier (RB, IN, ...)	Dependent: Ignored (DT, TO, ...)
	Core **Event**	nsubj (*arg0*) nsubjpass (*arg1*/*arg2*) dobj (*arg1*) iobj (*arg2*)	xcomp (*argctl*) ccomp (*argctl*) acomp (*argctl*)		
	Non-core **Event**	pobj (prep) tmod	pcomp (prep) advcl (mark) parataxis (*next*) conj (cc/*and*)	neg prep mark advmod	aux cop
	Nominal	pobj (prep) poss (*poss*) conj (cc/*and*) nn	pcomp (prep) amod (*arg-of*) rcmod (*arg-of*) infmod (*arg-of*) partmod (*arg-of*)	neg prep num	det predet

Table 5.1: The dependencies currently supported by our approach, grouped by the expected types of their arguments, and their respective translations. *arg0/1/2/ctl* translations assign values to the head's respective core argument. *arg-of* translations are similar, however the dependency direction is reversed. Other dependencies create *modifiers*. Where a value is given in brackets, that denotes the value of the modifier node which connects the head to the dependent. For dependencies without bracketed values, the value of the dependent itself is the value of the modifier.

nsubj (nominal subject) dependency expects that its head should always be an event and its dependent should always be a nominal. This type check helps us to identify and resolve many POS-tagging and parser errors.

Many dependencies such as nsubj are simple cases in which the knowledge graph exactly mirrors the dependency structure, just with a different label. However this is not always the case: the direction of edges may need to be reversed (e.g. for an rcmod (relative clause) dependency), additional nodes and/or edges not present in the dependency structure may need to be added (e.g for a conj (conjunction) dependency), or the head node may need to be switched in order to satisfy type constraints, which corresponds to shifting the position of a node in the graph structure (e.g. for a mark (marker) dependency). Table 5.1 classifies each supported dependency according to the expected types of its head and dependent, and the *translation* which is applied. We provide further details of the translations for some of the more complex cases in Section 5.3.1.

One important feature of our approach is that we do not binarise our grammar like most sim-

ilar approaches[3]. One result of this is that it is possible for one constituent to have multiple dependencies. The final step therefore involves *combining* all dependencies of a constituent. The most important thing to note is that the head daughter is responsible for the semantics of the constituent, so the head daughter's literal is always passed up directly. Then, we consider all of the modifications that need to be made to the head, according to the identified translations. How to combine the translation depends on whether they make changes to the same literal, or different ones, as demonstrated by the following examples.

Example 5.7. (Combining dependencies: non-core arguments) Consider the verb phrase construction in the sentence "Jim (VP (VBD[head] yelled) (PP[prep] at Kevin) (SBAR[advcl] because he was so upset))". Here both of the dependents are modifiers, so each creates its own rule as they do not need to change the head (the *yell* event).

```
vp -> vbd pp sbar {
  modifier(ID, Lemma, Arg0, Arg1)
    :- modifier(ID, Lemma, Arg0, null)@2,
       event(Arg1, _, _, _, _, _)@1.
  modifier(ID, Lemma, Arg0, Arg1)
    :- modifier(ID, Lemma, Arg0, null)@3,
       event(Arg1, _, _, _, _, _)@1.
  ...
}
```

Example 5.8. (Combining dependencies: core arguments) Consider the verb phrase construction in the sentence "Jane (VP (VBD[head] gave) (NP[iobj] Joan) (NP[dobj] candy))". Here each of the noun phrases sets one of the *give* event's core arguments. Therefore the dependencies must be combined to form a single rule.

```
vp -> vbd np np {
  event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)
    :- modifier(ID, Lemma, Arg0, null, null, ArgCtl)@1,
       nominal_unused(Arg1, _)@3,
       nominal_unused(Arg2, _)@2.
  ...
}
```

Composition Once the lexicon and grammar rules have been produced, a sentence's knowledge graph structure can be extracted simply by parsing the sentence and checking the atoms which hold at the root. Figure 5.2 shows how the grammar rules compose the semantics for our running example, "Jim yelled at Kevin because he was so upset." Note how the root node produces the correct predicate structure, matching what we described in Example 5.1 and thus corresponding to the knowledge graph in Example 5.2.

[3]There are two reasons why such a step is not necessary in our approach. Firstly, although a dependency structure alone imposes no ordering on dependencies such as `nsubj` and `dobj`, we use a constituency grammar as our ASG's base, which encodes exactly such an ordering. Secondly, even when the constituency grammar does not specify the ordering (for example, in the case of `dobj` and `iobj` dependencies), annotated atoms in an ASG can specify their child's index, which gives us the flexibility to handle different possible orderings of constituents, unlike an approach which uses lambda calculus.

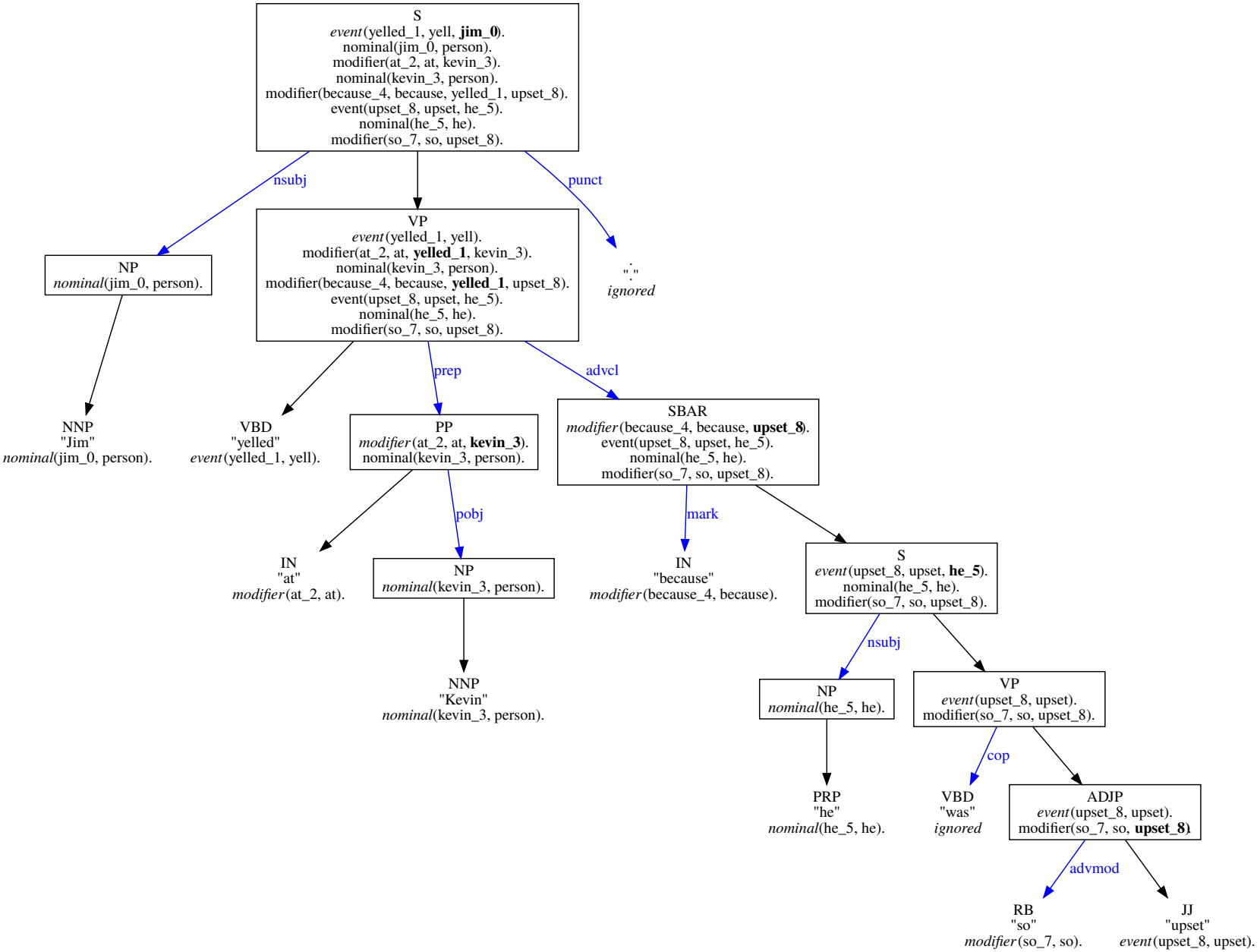


Figure 5.2: A graphical representation of the parse tree generated for the sentence “Jim yelled at Kevin because he was so upset.” All the atoms which hold at each node are shown. The italicised predicate at each node corresponds to the “head concept” and gives the node’s type. Changes between a literal in a node and its child are shown in bold. Placeholder arguments are omitted for brevity.

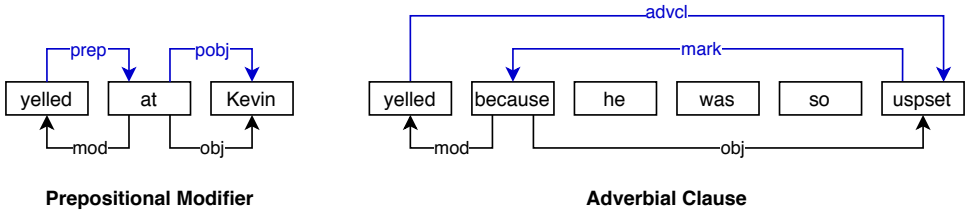


Figure 5.3: Comparing our semantic structure (black) to dependency structure (blue).

5.3.1 Analysis of Some Linguistic Constructions

As we have seen, in many cases, dependencies can be translated to our semantic representation without too much difficulty: core arguments for events are handled by direct assignment, and non-core arguments are handled by setting arguments of modifiers. We now discuss some linguistic constructions which require more complex approaches.

Prepositional Phrases and Adverbial Clauses Adverbial clauses and prepositional phrases, which may modify events or nominals with another event or nominal, are both handled in a similar way. Let's again consider the simple prepositional phrase modifier "(VP (VBD yelled) (PP[prep] (IN at) (NP[pobj] (NNP Kevin))))". We note that the PP is correctly assigned the *modifier* type, since its head has the IN (preposition) tag, which generates a *modifier*. The `pobj` dependency therefore just needs to set the modifier's object:

```
pp -> in np {
  modifier(ID, Lemma, Arg0, Arg1)
    :- modifier(ID, Lemma, Arg0, null)@1,
       nominal(Arg1, _)@2.
  ...
}
```

A `pcomp` dependency, in which the preposition's object is an event, is handled in the same way. When we reach the verb phrase we then set the modifier's subject to be the *yell* event (see the first rule of Example 5.7).

The difficulty with adverbial clauses is that the dependent type is usually an *event*, rather than the *modifier* we want. That is because, as shown in Figure 5.3, the dependency structure is quite different from our proposed structure. For example, in the phrase "(SBAR (IN because) (S he was so upset))", the *upset* event is considered the head, but our representation calls for subordinate clauses such as this one to be represented by a modifier. This difference in structure is resolved simply by treating `mark` dependents as heads in order to satisfy the type requirements.

Negation Negation is handled in a similar way to any other simple modifier. Negation will produce a *neg* dependency between the negating word and the event, nominal or modifier

being negated, which we translate directly into our own semantic representation. Let us consider as an example the sentence

*The trophy **does not fit**.*

Our translation approach gives a logical representation of the form

$$nominal(n1, trophy) \wedge event(e1, fit, n1) \wedge modifier(m1, neg, e1)$$

There is one clear issue with this representation, which is that it entails

$$nominal(n1, trophy) \wedge event(e1, fit, n1)$$

As such, we make one change compared to standard modifiers, unique to negation, which is that we have a post-processing step in order to change the predicate name for negated literals. This ensures that the representation for, say, negated events and non-negated events is different. The step is implemented by the ASG's `start` rule and takes the form:

```
start -> s {
  neg_event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)
    :- event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)@1,
       modifier(_, neg, ID)@1.
  event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)
    :- event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)@1,
       not neg_event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl).
  ...
}
```

Coordination Coordination is a tricky construction to handle, as it effectively results in several head literals. Let us consider, for example, the following case of NP-conjunction

***Sam and Amy** are passionately in love.*

We aim to generate a logical form with *distributive reading*, i.e. we wish for this sentence to be interpreted as *Sam is passionately in love* and *Amy is passionately in love.* This requires that the noun phrase *Sam and Amy* be treated as having two head literals: `nominal(sam, person)` and `nominal(amy, person)`.

Many approaches which rely on lambda calculus for composing semantics require post-processing to fix the semantics of sentences with coordination due to the difficulty of supporting multiple heads in this way. Our approach, on the other hand, handles such constructions quite naturally:

```
np -> nnp cc nnp {
  nominal_unused(ID, Lemma) :- nominal_unused(ID, Lemma)@1.
  nominal_unused(ID, Lemma) :- nominal_unused(ID, Lemma)@3.
  modifer(ID, Lemma, Arg0, Arg1)
    :- modifier(ID, Lemma, null, null)@2,
       nominal(Arg0, _)@1,
       nominal(Arg1, _)@3.
  ...
}
```

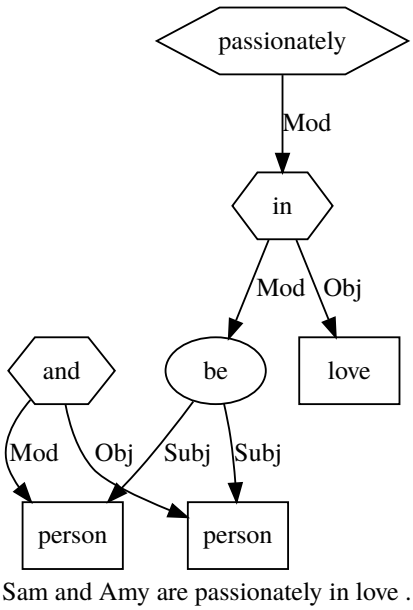


Figure 5.4: A produced knowledge graph demonstrating the handling of coordination.

Since the head of a constituent is determined simply by a body atom specifying a literal of a certain type at a certain child (which has not yet been finalised), we can simply produce two such literals, and any ASP rules in production rules applied further up the parse tree automatically will automatically apply to both of them, producing the distributive reading we desire. In the production rule shown, the first two ASP rules implement this idea. We note that approach has the benefit of requiring no further modification to our ASG whatsoever, only to the intermediate representation we use to generate ASG rules (which did not originally support multiple heads).

Let us also consider the third ASP rule in the production rule shown previously. We chose to add a modifier that links the two conjuncts by their coordinating conjunction. This is because coordinating conjunctions may in themselves hold important semantic information: for example *so* might be used to provide motivation, or *but* to highlight contrast. The final produced representation is shown in Figure 5.4.

Although we have just considered the case of NP-conjunction here, other kinds of coordination are handled in exactly the same way.

Passives Consider the passive construction:

The drain ***is clogged*** *with hair.*

Examples such as this one require special handling since the nominal which takes the place as the syntactic subject of the *is clogged* construction is really its semantic object of the *clogged* event, not its semantic subject.

In order to be able to differentiate between the two possible cases (whether subject dependency really acts as a subject, or rather as an object), we need to change the representation

of the clogged event. This is done by changing the predicate name when a passive construction is identified, from `event` to `event_passive`. These passive forms can be identified by an `auxpass` dependency; in our example it's between *clogged* and *is*, and it would generate a production rule of the form:

```
vp -> vbz vp {
  event_passive(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)
    :- event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)@2,
       event(_, be, _, _, _, _)@1.
  ...
}
```

In a nominal subject rule, such as the one shown in Example 5.5, we then simply add a different rule for `event_passive` predicates, which sets the object argument instead of the subject argument. At the root node, a final rule converts passive events back into standard events in order to maintain the usual predicate structure.

Relative Clauses Relative clauses such as "which ran up the tree" in the sentence

The dog chased the cat, ***which ran up the tree****.*

are the first of a tricky set of constructions involving *non-local dependencies* which we will consider. The difficulty arises since the semantic subject of the *running* event here is the *cat*, which never appears as a syntactic argument of *ran*. In CCG-based approaches, these cases often require special handling in the form of co-indexed categories: the lexical entry for *which* would be responsible for linking the object of *chased* to the subject of *ran*. Meanwhile, dependency-based approaches must augment the dependency structure in order to add an additional dependency, which adds an equality constraint between the two nominals.

Our approach is different in that we do not treat these as two separate nominals which are forced to become equal, but instead leave the subject of *ran* unspecified until we have its real value. As such, the lexical entry for the wh-pronoun *which* is marked as unfinalised, allowing it to be replaced when a nominal with a `rcmod` (relative-clause) dependent is reached. The production rule which handles the `rcmod` dependency would thus take the form:

```
np -> np "," sbar {
  nominal_unused(ID, Lemma)@1 :- nominal_unused(ID, Lemma)@1.
  event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)
    :- event(ID, Lemma, unfinalised, Arg1, Arg2, ArgCtl)@3,
       nominal_unused(Arg0, _)@1.
  ...
}
```

Note how this reverses the dependency structure: while the `rcmod` dependency points from the nominal to the event, we mark the nominal as an argument *of* the event. Our produced representation is shown in Figure 5.5.

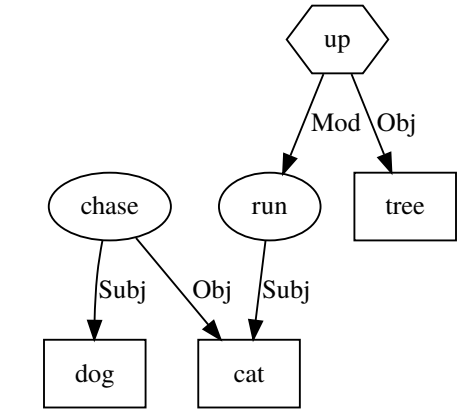


Figure 5.5: A produced knowledge graph demonstrating the handling of relative clauses.

It is important to note that relative pronouns can take the place of an object or a subject in the relative clause. These cases are handled automatically since the unfinalised argument's position is decided by an `nsubj` or `dobj` dependency within the relative clause. Furthermore, relative pronouns can be omitted, but only when they are objects of the clause. This case is handled by replacing the object when we find an `rcmod` dependency without any unfinalised arguments.

Control and Raising The constructions of *control* and *raising* occur frequently in the WSC: there are 95 instances of these constructions among its 273 problems.

Control is a construction in which the semantic subject of a predicate is determined by some expression in context (Bhatt, 2006). For example, consider the sentence:

$$\textit{John}_i \textit{ promised Bill } [\textit{PRO}_i \textit{ to leave}]. \tag{5.1}$$

Here *John* is the agent of both *promised* and *leave*, event though he is the syntactic subject for only *promised*. This is called subject control.

$$\textit{John ordered Bill}_i \ [\textit{PRO}_i \textit{ to leave}].$$

Here Bill is the patient of *ordered* and the agent of *leave*, even though he is the syntactic object for only *ordered*. This is called object control.

Raising, on the other hand, is a construction in which an argument which belongs semantically to a subordinate clause is a syntactic argument of the main clause, i.e. instead of sharing an argument, the argument *moves* (Bhatt, 2006). Let us consider an example:

$$\textit{Rebecca}_i \textit{ seems } [t_i \textit{ to suspect something}]. \tag{5.2}$$

Here, *seems* is a raising-to-subject verb. We note that *Rebecca* is the agent of *suspect* and not *seems*, even though she is the syntactic subject of *seems* and not *suspect*. A second type of raising verb is a raising-to-object verb such as *believed*:

$$\textit{Carol believes Rebecca}_i \ [t_i \textit{ to suspect something}].$$

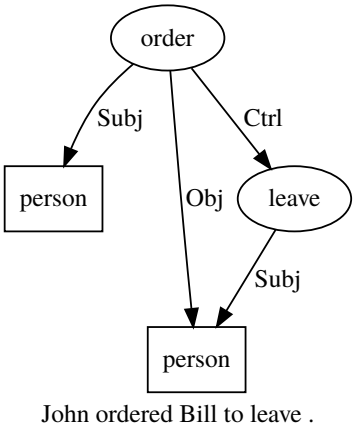


Figure 5.6: A produced knowledge graph demonstrating the handling of control.

Here we note that Rebecca is the agent of *suspect* and not the patient of *believes*, although she is the syntactic object of *believes* and not the syntactic subject of *believes*.

We choose to treat both constructs identically. An example of their desired representation is shown in Figure 5.6. Firstly, both cases use the "control argument" to link the control/raising verb to the infinitival. In both cases, we also attempt to determine the subject of the infinitival, instead of leaving it unspecified, as it can be done with a high level of accuracy and significantly improves the quality of the semantic representation. One thing to note is that this means that our approach would, for example, leave *Rebecca* as the subject of *seems* (in addition to making her the subject of *suspect*) for (5.2). Although this treatment is technically incorrect, it is widely-used in semantic parsers (Reddy, 2017) and significantly simplifies our approach. In practice, raising occurs much less frequently than control, and we are yet to find any example where the additional argument leads to issues in learning.

However, these constructs still pose a significant challenge for our approach due to the misalignment between syntactical and semantic arguments, and also a lack of type information to determine between the subject and object cases. One method for handling these cases is to rely on the lexicon, in the same way that a CCG-based approach would. It is widely accepted that whether a control construction is an instance of subject or object control depends solely on the control verb (Bhatt, 2006), so the only necessary addition is that the lexical entry for the control verb must specify whether it is controlled by a subject or an object. The unassigned subject of the infinitival can then be assigned at the same time as the specified argument of the control verb.

However, we were unable to find an existing resource that offers a complete classification of control verbs into subject and object control verbs, and found machine learning techniques based on semantic role labelling to be unreliable. We therefore use an approximation based on the *minimal distance principle* (MDP) (Bhatt, 2006). Since there are many object control predicates but few subject control predicates, and even fewer subject control predicates which also select an object (like *promised* does in (5.1)), the approach of simply selecting the closest noun to infinitival (the object of the control verb if there is one, otherwise the subject) to be the infinitival's subject works surprisingly well. Indeed, of the 95 instances of control and raising in the WSC, there is just one that violates the MDP, which is the example shown in

(5.1).

The MDP can also be implemented very elegantly by our ASG. When we see an `xcomp` dependency (a clausal complement without a subject), we simply mark the subject as *unfinalised*. `nsubj`, `dobj` and `iobj` (nominal subject/object) dependencies are then made to always replace an *unfinalised* subject, so that the subject is always assigned to be the closest argument above it in the parse tree. For example, the nominal subject construction from Example 5.5 would be updated with the additional rule:

```
s -> np vp {
  event(ID, Lemma, Arg0, Arg1, Arg2, ArgCtl)
    :- event(ID, Lemma, unfinalised, Arg1, Arg2, ArgCtl)@2,
       nominal_unused(Arg0, _)@1.
  ...
}
```

Ignored Constructions We currently ignore the *tense* of events and *quantification* of nominals, although tense could be handled with a simple extension to our representation and lexicon, and quantification could be handled by adding modifiers for determiners and a post-processing step to introduce universal quantification where necessary. However, there are no examples where quantification is relevant in the WSC, and only a single schema-pair which is affected by the omission of tense information:

> Fred is the only man still alive who remembers my great-grandfather.
> He [*is*/*was*] a remarkable man. Who [*is*/*was*] a remarkable man?
>
> *Answer 0*: Tom
> *Answer 1*: Ray

Since we cannot differentiate between verbs in different tenses, our representation of both schemas is identical; thus unfortunately we are forced to answer one of these schemas incorrectly if the other is answered correctly (assuming the same examples are chosen).

5.3.2 Background Knowledge Generation

Once we have generated an ASG that can act as a domain specific semantic parser, we then search for relevant background knowledge and augment the ASG in order to be able to capture synonymy, entailment, and similarity of phrases which might have originally produced different representations. Although we want to augment the ASG with as much semantic information as possible, it's vital that the annotations are accurate. Otherwise, there is a chance that background knowledge may spuriously make irrelevant examples relevant, and thus make a learning task significantly more difficult, or even unsatisfiable. In other words, we have a slight preference for precision over recall.

We considered a number of lexical and commonsense knowledge bases, including WordNet, VerbNet, FrameNet, PropBank, ConceptNet, WebChild and ATOMIC. In this project, we

choose to focus on WordNet. Its focus on lexical semantics makes it highly structured, and it has been produced by hand by lexicographers, making it significantly more accurate when compared to knowledge bases that are built through crowd-sourcing or automated knowledge extraction methods.

WordNet The background knowledge generation process is applied per token. Let us consider an example.

*Bob **paid** for Charlie's college education. He was grateful.* (5.3)

The token *paid* has the lemma *pay* and a verb POS tag. Looking this up in WordNet, we find that there are 11 possible senses for the verb *pay*. Generating background knowledge for all of these entries is likely to result in a huge volume of mostly irrelevant knowledge, so we first use a word sense disambiguation model to select a single sense. In this case, *paid* is assigned the most common sense, `pay.v.01` ("give money, usually in exchange for goods or services"). The first rule produced is thus

$$event(paid\_1, pay\_v\_01, Arg) \leftarrow event(paid\_1, pay, Arg).$$

which links the lemma *pay* to its sense. This automatically captures some synonymy, as other lemmas may be assigned to the same sense and thus produce matching *event* literals. The next step is to add knowledge from hypernymy and entailment. For example, pay is marked as a hyponym of the sense `give.v.03`, so we additionally produce the rule

$$event(ID, give\_v\_03, Arg) \leftarrow event(ID, pay\_v\_01, Arg).$$

To more concretely demonstrate the utility of this background knowledge, let us consider another example:

*Katrina **bought** some catnip for Maria's brand new kitten,*
which made Maria feel very grateful. (5.4)

Here *bought* is assigned the sense `buy.v.01`, which in WordNet, is marked to entail both `choose.v.01` and `pay.v.01`. Note how the addition of background knowledge therefore results in the literal $event(pay\_1, pay\_v\_01, bob\_0)$ holding for (5.3) and $event(bought\_1, pay\_v\_01, katrina\_0)$ holding for (5.4), thus allowing a single rule with a body atom

$$event(PayEvent, pay\_v\_01, Payer)$$

to cover both examples.

ConceptNet We also add a small amount of background knowledge from ConceptNet. ConceptNet defines some 40 kinds of relations between words and phrases. Currently we use just one: the *HasProperty* relationship. The reasoning behind this choice is that most other relations are (a) defined by WordNet (e.g. *IsA*, *PartOf*) or (b) express the commonsense concepts that we are intending to learn (e.g *CapableOf*, *UsedFor*).

ConceptNet is more challenging to work with than WordNet, since it does not include word senses and thus often returns spurious results. For example, the word *ball* has the properties *round* and *spherical*, but also *formal*. Furthermore, we find that ConceptNet can often return results that are never useful — for example, *steel* has the property *strong_and_hard*, yet no other word in ConceptNet shares this property.

To provide an example of where how we use ConceptNet's *HasProperty* relation, let us consider the schema:

The large ball crashed right through the table because it was made of ***steel****.*

In this schema, the nominal *steel* is associated with the property *hard*, so the following rule is generated:

$$modifier(conceptnet1, hard, ID) \leftarrow nominal(ID, steel).$$

Now consider an example of the form:

A stout tree trunk made of ***stones*** *and cement*
— has fallen and crashed through the tomb's surface.

Here the word stones is also assigned the property hard, producing the rule:

$$modifier(conceptnet2, hard, ID) \leftarrow nominal(ID, stones).$$

thus allowing us to specify that thing which *is made of a hard material* is the thing crashes through another object.

Chapter 6

Learning Commonsense Knowledge

In this chapter, we discuss our approach to learning commonsense knowledge on top of our automatically-generated base ASGs. There are two key stages to our approach: the first is a *knowledge hunting* stage in which we seek to find and comprehend examples which are related to the problem being solved; and the second is a *learning* stage in which we seek to form a hypothesis that explains our understanding of those examples. We can then take this learned hypothesis and apply it to the problem at hand.

This approach is motivated, in part, by how people are thought to learn commonsense knowledge. A key part of human learning is *learning by language* (Levesque, 2017). For example, Levesque (2017) notes that our understanding that *bears hibernate* likely comes from *reading* and not direct experience. Our *knowledge hunting* stage emulates recalling knowledge that one might have heard or read before. In some cases, we might simply learn concepts by direct instruction: that is, we might read directly that *bears hibernate*. In our approach, such cases would be imitated by the presence of existing background knowledge. However, in many other cases, we have never encountered the knowledge directly, but we are able to make analogies to things that we do know about: this is what our *learning* stage attempts to do. For example, while we may never have been told directly that *something that is hibernating is asleep*, we may have *learned* to associate the two concepts by encountering examples where a hibernating animal sleeps, while we know that animals do not always sleep in general.

6.1 Knowledge Hunting

Firstly, we discuss our approach to *knowledge hunting*. The broad goal of this stage is to find texts that express some commonsense knowledge, in an unambiguous way, that is relevant to the problem at hand. To motivate the process, let us consider again the Winograd schema:

> Jim *yelled at* Kevin because **he** was so upset. Who was so upset?
>
> *Answer 0*: Jim
> *Answer 1*: Kevin

The kind of text that we might hope to find for this schema is:

> **Example 6.1.** **An ex** yelled at me because **she** got upset at how much we disagreed on how I should handle a hypothetical situation.

We note that in this case, the pronoun *she* is unambiguous. There are two candidate referents, *an ex* or *me*, but *an ex* is the only valid referent since *she* is a third person pronoun. This text is useful because it is an example that demonstrates that *the person who yells is the person who is upset*, which is the correct hypothesis to be made in this case.

Knowledge Hunting as a Ranking Problem In the existing literature, knowledge hunting is treated as a problem of information retrieval. As we discuss in Section 3.2, the goal of these approaches, given a coreferencing problem, is to generate a query (usually taking the form of a regular expression) that finds texts expressing knowledge that is relevant for solving that problem. We slightly reframe this view, splitting our approach into two stages, and treating knowledge hunting mainly as a ranking problem. The first stage sources a (possibly very large) set of relevant texts (the information retrieval part), while the second determines which of those texts are most relevant (the ranking part). Separating the retrieval of possibly relevant texts from the process to determine their relevance allows us to both retrieve more texts for consideration (hopefully improving recall) and rely on far more sophisticated methods for determining relevance than just simple regular expressions (hopefully improving precision). In this section, we assume a set of candidate texts exists and discuss only our approach for determining relevance. We consider the sourcing of the candidate texts to be an implementation detail, and discuss it in Section 7.2.2.

Goals of Knowledge Hunting In previous approaches to knowledge hunting the goal is purely to find examples that help you to determine the correct answer, like Example 6.1. This means you would want to rule out an example of the form:

> **Example 6.2.** Anyway the mum went mad at **my son** and shouted at **him**. **He** was upset - **he** is only 5 - and **he** ran upstairs.

In this example, the pronoun *he* is again unambiguous, this time because of its gender, but this text, unlike Example 6.1, seems to suggest that *the person who is shouted at is the person who is upset*. Clearly this is not what we want to learn in order to solve the original problem, but we argue that the example is still relevant. In Example 6.1, the connective *because* signifies that there is some causality — specifically, somebody being upset *causes* them to yell. Meanwhile Example 6.2 has no such construction — here, the fact that the son being upset follows the son being shouted at in the text instead signifies causality in the other direction, specifically somebody being shouted at *is followed by* them becoming upset. Example 6.2 is still relevant because it tells us that the hypothesis that *the person who yells is always the person who is upset* is not a good hypothesis (it is too general), and forces us to take into account the causality expressed in Example 6.1. This motivates the first goal of our knowledge hunting approach:

1. *Any* text that is able to relate either of the candidate referents with the pronoun should be determined to be relevant.

However, our learning approach does not scale well with many examples (especially when those examples are complex and feature many different linguistic constructions), necessitating a limit on the number of examples selected for consideration by the learner. If we have to choose between Examples 6.1 and 6.2, clearly we would prefer Example 6.1, since that is the text which gives us the correct answer. To demonstrate more clearly that our first goal is not sufficient on its own, let us consider a slightly more complex Winograd schema:

> Tom gave Ralph a lift to school so **he** wouldn't have to *drive alone*.
>
> *Answer 0*: Tom
> *Answer 1*: Ralph

and the example:

> **Example 6.3.** I gave **Juli** my keys so **she** could drive.

This example satisfies our first goal, since we could relate the person who drives to the person who is given something. There are many similar, short and simple examples that will only serve to teach the learner nonsense. But the example

> **Example 6.4.** The couple first met in 1977, when **Ralph** — driving the same Beaumont — gave Cheryl a lift to Canada celebrations at Wascana Centre.

is clearly a lot more relevant, because it talks about giving somebody *a lift*. This demonstrates two things. Firstly, simple features such as length or syntactic similarity are not, on their own, a good indicator of whether the knowledge text is relevant[1]. Secondly, matching as much as possible of the context from the original problem is vital, in order to provide the learner an opportunity to work out which context is important. Thus the second goal of our knowledge hunting approach is that:

2. The text which shares more important context with the problem should be preferred.

[1]This example additionally provides motivation as to why using existing machine learning approaches for semantic textual similarity may not be the ideal approach. The first example is syntactically similar to the problem, and the main concept of the sentence is *giving* (the keys to Juli) — the same as in the problem. Meanwhile, the second sentence is syntactically quite different and the main concept is (the couple) *meeting*. There is also a lot of irrelevant context in the second example which might lead to a semantic textual similarity model assigning it very low similarity score. Indeed, when representing the sentences with Google's Universal Sentence Encoder (Cer et al., 2018), and comparing them with the schema's representation, we found that the cosine distance for Example 6.4 was greater than for Example 6.3 (i.e. Example 6.3 was determined to be more similar). Meanwhile, a more specialised approach can focus in on the nominals "matching" the candidates and the pronoun from the problem and ignore the rest of the sentence entirely.

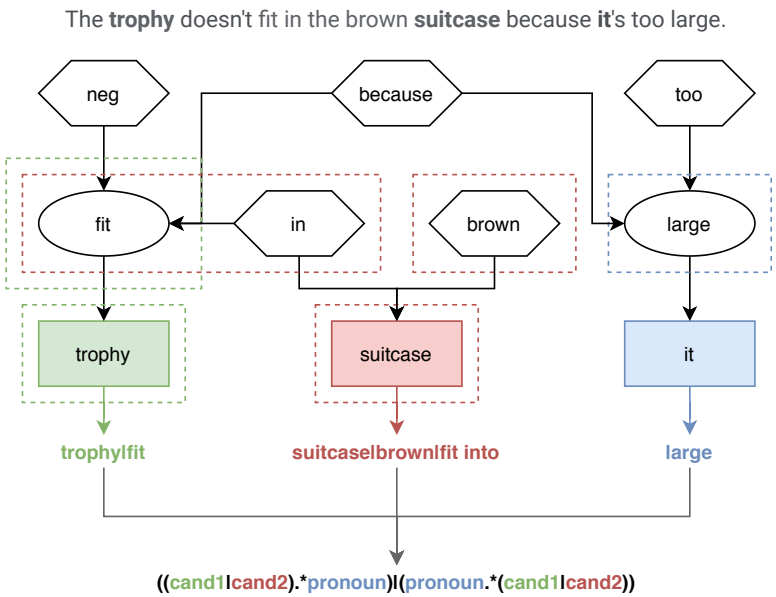


Figure 6.1: An overview of our approach for constructing a query for initial filtering from a Winograd schema's knowledge graph. Dotted boxes in green show the nodes used to generate queries for the target referent, red for the other candidate and blue for the pronoun. The queries are simplified here for brevity.

6.1.1 Initial Filtering

Our knowledge hunting process is computationally expensive as it is based on semantics and thus requires a full semantic parse of every example to be considered. Therefore, we first have a stage to retrieve all possibly relevant examples, and discard the majority of texts, which are completely unrelated. The main aim here is to achieve perfect recall: we should not discard any texts that could be relevant.

Preprocessing To allow ourselves to easily capture different forms of the same word (e.g. different conjugations of a verb), synonymy and certain grammatical structures without building complex queries, all examples are first preprocessed. Specifically, we tokenise each example and determine the part-of-speech tag, lemma and WordNet sense for each token. This step is performed just once, and the results over the entire set of examples are saved for all future queries to use.

Query Generation The filtering is then performed by executing a query over the preprocessed examples. To demonstrate how the query is constructed, let us consider an example. The knowledge graph for "the trophy didn't fit in the suitcase because it was too large" is shown in Figure 6.1. This query is constructed by considering the knowledge graph as follows:

1. We firstly identify the two candidate referents and the pronoun. Our approach to this is rather rudimentary and is described in Subsection 7.2.1. In Figure 6.1, we show the two candidates in red and green and the pronoun in blue.

2. We then consider each of these nominals in turn, generating a set of queries which describe minimally the context of that nominal.

   (a) If the candidate is not a proper noun, then a query is generated based on the candidate itself. E.g. for "suitcase" a query is generated which will match either the lemma, (`suitcase`) or its sense (`bag.n.06`) in the preprocessed text.

   (b) For each node in the semantic graph of which the nominal is an argument, an additional query is generated. E.g. for "suitcase" a query is generated to match the lemma or sense of the word *brown*.

   There are two special cases here. Firstly, if the nominal is the object of a modifier, then the query is based on both the modifier and the entity being modified (to avoid spurious matches on prepositions alone). For our suitcase example, this means generating a query matching any word with the lemma or sense of *fit* followed by any word with the lemma or sense of *in*. Secondly, if the modifier is a *possession* modifier, then the query is instead based on the word's part-of-speech: specifically, we are looking for a `PRP$` (possessive pronoun: *my*, *your*, *his*, *her* etc.) or a `POS` (possessive ending: *'s*) tag.

   Note how, motivated by our goal to achieve good recall, we do not explore deep into the knowledge graph, considering only the nodes that are directly connected to the nominal being considered.

3. For each nominal, a single query is constructed by allowing any of the individual queries from the previous step to match. For our suitcase example, we therefore would build a query of the form

$$\begin{aligned} & (lemma = suitcase | sense = bag.n.06) \\ \textit{or} \quad & (lemma = brown | sense = brown.a.00) \\ \textit{or} \quad & ((lemma = fit | sense = fit.v.02)\ \textit{followed by}\ (lemma = in | sense = in)) \end{aligned}$$

   The reason to allow any of these queries to be matched is that any of them could be the relevant part — for example, in this schema it is clearly the event *fit* that is most important, but in the schema

   ***Frank** felt vindicated when his longtime rival **Bill** revealed that*
   ***he** was the winner of the competition.*

   clearly the modifier *rival* is important, while in a schema like

   *Sam tried to paint a picture of **shepherds** with **sheep**, but*
   ***they** ended up looking more like golfers.*

   clearly the identity of the nominals *shepherds* and *sheep* themselves are important.

4. Finally a query is generated to match any sentence which has some matching context for the pronoun and some matching context for *either* of the candidates. I.e. the query takes the form `(pronoun.*(cand1|cand2))|((cand1|cand2).*pronoun)`. This is motivated by our goal to consider *any* relevant text — there is still something to be learned from a text which accepts or excludes just one of the candidates.

This final query is then compiled and checked against every example.

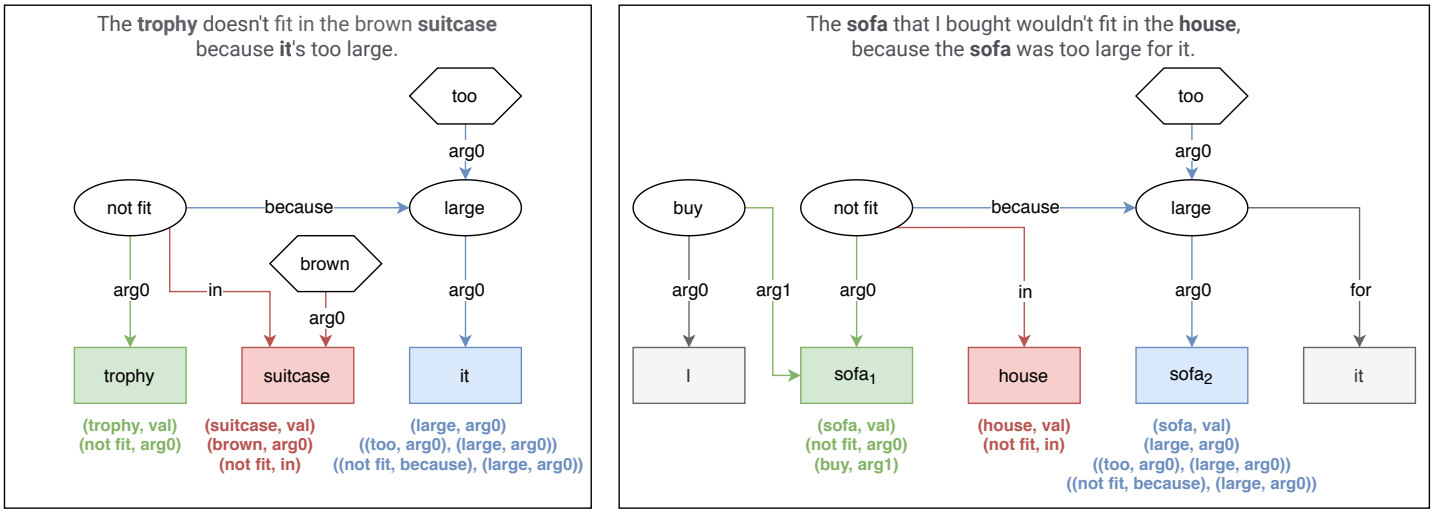


Figure 6.2: An overview of how we define the *context* function for a node. We preprocess the graph to "collapse" modifiers and then find all paths to the node by following the edges backwards.

6.1.2 Relevance Scoring

Our chosen learning algorithm is limited in the number of examples it may take. Therefore, after reducing the set of candidate knowledge texts to the ones which could be relevant, we then attempt to determine which ones are actually the *most* relevant.

Fitness Function To determine which examples are most relevant, we propose a fitness function that is motivated by previous work on relevance theory in knowledge graphs (Schüller, 2014). Schüller (2014) finds that the single most helpful heuristic is to do with how many nodes from the background knowledge are "combined" with nodes from the Winograd schema. We come to a similar conclusion, although our method for calculating a value for such a heuristic is quite different to his proposal.

To determine how much context is shared, we first define a function, *context*, to return a set of paths defining, as fully as possible, the context of a nominal candidate. Figure 6.2 shows an example of how this is done. Firstly, we collapse modifiers in the original knowledge graph. Specifically, modifiers which link two concepts are collapsed into a single edge from the modified entity to the entity that modifies it. This gives us a rough argument structure view of the text. We then start from each nominal node and follow the incoming edges recursively, building up a list of paths to each nominal. As shown in Figure 6.2, for our ongoing example, this gives us the following sets of paths[2]:

[2]For brevity, we ignore senses here and look only at lemmas. In reality, as was the case for initial filtering, these paths need to encode both the lemma and the sense of each node to allow matching of synonyms and entailed concepts.

$$
\begin{aligned}
context(\text{trophy}) &= \{(trophy, arg0), (not\ fit, arg0)\} \\
context(\text{suitcase}) &= \{(suitcase, arg0), (brown, arg0), (not\ fit, in)\} \\
context(\text{it}) &= \{(large0, arg0), ((too, arg0), (large, arg0)), ((not\ fit, because), (large, arg0))\}
\end{aligned}
$$

Given an example with a labelled coreference, such as the sentence shown in the left-hand-side of Figure 6.2, we perform the same process on the example and calculate an intersection-over-union score. For the example in Figure 6.2, this would be

$$
\begin{aligned}
IOU(knowledge, schema) =& \frac{|context(\text{sofa}_1) \cap context(\text{trophy})|}{|context(\text{sofa}_1) \cup context(\text{trophy})|} \\
&+ \frac{|context(\text{house}) \cap context(\text{suitcase})|}{|context(\text{house}) \cup context(\text{suitcase})|} \\
&+ \frac{|context(\text{sofa}_2) \cap context(\text{it})|}{|context(\text{sofa}_2) \cup context(\text{it})|} \\
=& \frac{1}{4} + \frac{1}{4} + \frac{3}{4} = 1.25
\end{aligned}
$$

The motivation behind this formulation is simply that we want to encourage shared context between the candidates in the schema and its matched nodes in the knowledge text, and discourage missing context from either side (which could easily result in the answer being swapped). Using an intersection-over-union score has the additional benefit of normalising the scores between a value of 0 and 3, since each Winograd schema has one pronoun and two candidates.

Schüller (2014) also proposes several other heuristics, such as adding costs for unmatched background nodes, the radius of graphs and others. We choose not to investigate these heuristics as they have already been shown to be much less useful. We investigated weighting based on path length (to give greater emphasis to "long chains" of matched nodes) but found it had no impact on results.

However, there are several other heuristics that we found to be useful, albeit much less so than our context-based IOU score. Theses include the source of the knowledge (e.g. an example which has been labelled by hand is preferable to an example which was labelled automatically), as well as whether the knowledge texts involves just one of, or both of the candidates (knowledge texts which form matches to both candidates are preferred), and the depth of the parse tree (shorter, less complex texts are far faster to parse and so favouring these examples speeds up the learning considerably). Our final fitness function for a knowledge text k with respect to a schema s is defined as follows:

$$
fitness(k, s) = (IOU(k, s), -parse\ tree\ depth(k))
$$

We prefer whichever knowledge text achieves the highest score: i.e. we first choose the knowledge which is determined to be the most contextually relevant, and if two trees have the same IOU score, we then choose the one with the smallest parse tree depth.

Example Selection Examples are then selected simply by choosing the top N knowledge texts according to the fitness function.

6.2 Learning Task Generation

We now discuss, given a Winograd schema, a set of relevant knowledge texts (found as described in Section 6.1), and a base ASG capable of representing the schema and texts (generated as described in Section 5.3), how to generate a learning task whose goal is to learn the commonsense concepts encoded in the knowledge texts.

The Role of *Learning* in our Approach It is important to note that our approach is *not* a simple case of taking an existing reasoning approach (e.g. (Sharma, 2019)) and attempting to learn a program to encode that reasoning process, in that we do *not* attempt to learn a generalised program for *how* to extract commonsense knowledge. Such an approach would firstly require us to have training examples labelled with the relevant commonsense knowledge or at least their answers (a privilege which the WSC does not afford us), would secondly constitute an extremely complex task that would need to be able to deal with hundreds of domain-general examples and generate an excessively large hypothesis, and would thirdly provide no real tangible benefit compared to just using the original reasoning approach.

Instead, we propose an approach for generating learning tasks to learn individual snippets of commonsense knowledge. This, in many ways, reflects existing reasoning-based approaches (Emami et al., 2018; Sharma, 2019), modifying just a small component of these approaches to allow us to *learn* a generalised rule encoding the snippet of commonsense knowledge (which also functions as an explanation), instead of just answering the question directly.

Since each question in the WSC requires its own unique snippet of commonsense knowledge, we generate a separate learning task *per schema*. Although in many cases it would be preferable to treat the schemas in pairs (the knowledge encoded in the second schema is usually at least somewhat similar to the first), we choose not to do so. As noted by Levesque et al. (2012), this would be a "cheap trick" and would lead us to overestimate our ability to work out the necessary commonsense knowledge for a single schema.

As we will see, the decision to learn individual snippets of commonsense knowledge considerably decreases the challenge for the learner, and in fact our learning tasks are remarkably simple, often amounting to just concept learning. A lot of the heavy lifting is done in previous steps: the *ASG* handles most of the challenge of natural language understanding, and the *knowledge hunting* is responsible for performing quite a bit of reasoning. The latter is necessary simply because the WSC is difficult: it is a domain-general task, and if the learner had to deal with irrelevant examples, it would need to explain an excessive number of examples before finally learning a suitable hypothesis, and furthermore would need to explore an excessively large hypothesis space. Another pre-processing task we take on to aid the learner is the *labelling of examples*, which we discuss in the next section. Clearly some form of labelling examples is necessary as the learner needs something to learn from. But the difficulty of sourcing training examples and, in particular, negative examples, necessitates that our labelling approach does a fair amount more work for the learner, as we will see.

Having done all of this pre-processing, a very simple reasoning process to match the candidates in the knowledge texts to the ones in the schema, followed by a simple statistical approach (Emami et al., 2018) or final reasoning step (given a manually selected example)

(Sharma, 2019) could be used to determine the answer straight away. But in our approach, the learner has a simple but critical task: it must learn *how to explain* the labelled knowledge texts. We therefore never provide the learner with any information about how candidates in the knowledge texts can be matched to the ones in the schema — but instead rely on the learner to learn an explanation that we hope will carry over to the schema.

6.2.1 Example Translation

The first step in generating our learning tasks is to translate the knowledge texts into positive and negative examples for learning. There are three observations which motivate our process for constructing examples:

1. **Necessity of negative examples.** Firstly, let us note the need for negative examples, in some form. Excluding all answer sets with certain qualities is what forces the learner to make its hypothesis more specific. Otherwise, the learner would always predict the shortest, most general hypothesis resulting in a satisfiable program that covers all the positive examples. Such a hypothesis is likely to be too general to distinguish between the two candidates without a very carefully-structured hypothesis space, and even more likely to correspond to a nonsensical explanation.

2. **Difficulty of sourcing negative examples.** In the context of the Winograd schema challenge, negative examples would need to take the form of syntactically correct English sentences that are semantically invalid, because they contradict some commonsense knowledge that a human reader might have. This represents a significant challenge as it is not clear how one might source such sentences. As we will see, even sentences which are semantically valid and encode relevant commonsense knowledge can be difficult to find. Furthermore, language is complex and it is not usually the case that you can take a semantically valid sentence and manipulate it in a mechanical way to make it semantically invalid. This motivates an approach where negative examples are constructed from positive examples, but annotated with context describing some interpretation that we know to be incorrect.

3. **Performance implications of negative examples.** Finally, although we have been discussing "negative examples" so far, it is important to note that we can force the learner to specify its hypothesis without negative examples in the traditional sense. Specifically, if an ASG is stratified, then a parse tree produces a unique answer set, and the semantics of a negative example (that *no* answer set may extend the example) can be expressed simply by an exclusion inside a positive example (to ensure that the single answer set does not extend the example) (Law et al., 2019). This is beneficial, since as noted by Law et al. (2019), "ILASP tasks with no negative examples tend to run faster than equivalent tasks with negative examples, so when it is possible to modify the representation to eliminate negative examples it is often advantageous to do so." This motivates an approach in which the ASG is stratified, thus ruling out constructions of the form

$$is\_same\_as(pronoun, cand1) \leftarrow \text{not } is\_same\_as(pronoun, cand2).$$
$$is\_same\_as(pronoun, cand2) \leftarrow \text{not } is\_same\_as(pronoun, cand1).$$

which would produce one answer set with $is\_same\_as(pronoun, cand1)$ and another with $is\_same\_as(pronoun, cand2)$.

Labelling Examples To demonstrate how we label examples, let us again consider the knowledge text from Figure 6.2, "the *sofa*$_1$ that I bought wouldn't fit in the *house*, because the *sofa*$_2$ was too large for it," with respect to the Winograd schema "the *trophy* doesn't fit in the *suitcase* because *it*'s too large." For this example we would aim to generate some context of the form:

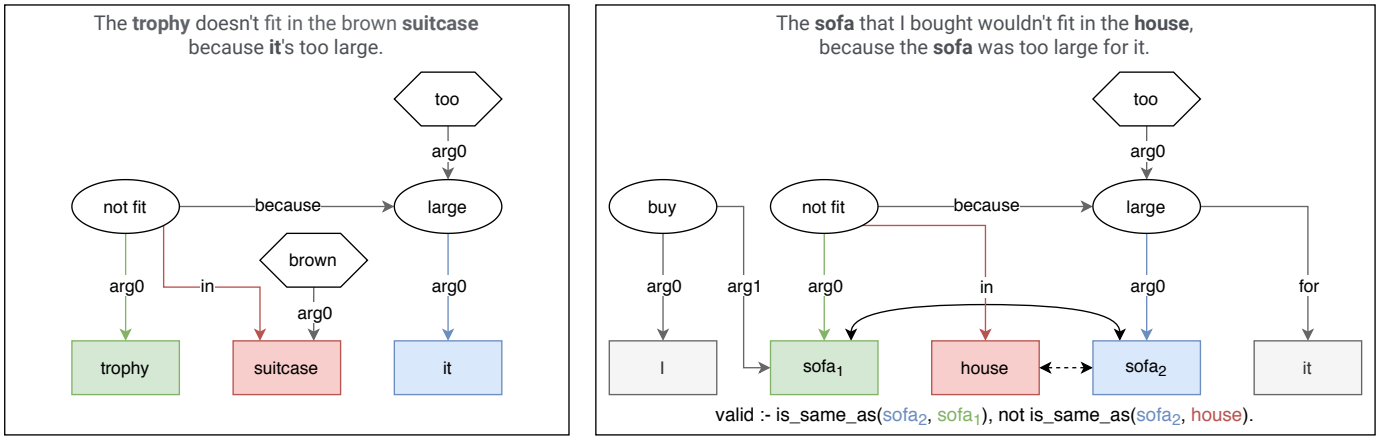
$$\begin{aligned} valid \leftarrow\ & is\_same\_as(sofa_2, sofa_1), \text{not } is\_same\_as(sofa_2, house). \\ \leftarrow\ & \text{not } valid. \end{aligned} \tag{6.1}$$

which serves to rule out any answer set in which either the two sofas have not been matched to the same entity, or in which the second sofa entity (corresponding to the pronoun in the Winograd schema) has been matched to the house.

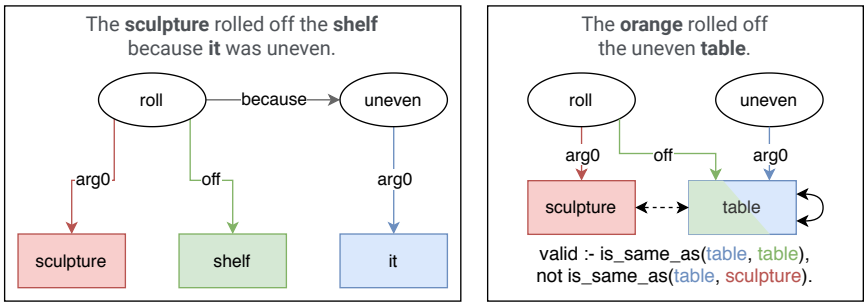
We note that this is equivalent to a formulation in which $\{is\_same\_as(sofa_2, sofa_1)\}$ and $\{is\_same\_as(sofa_2, house)\}$ are the inclusions and exclusions of the example respectively (i.e. the constraint in (6.1) is satisfied if and only if there is an answer set A with $\{is\_same\_as(sofa_2, sofa_1)\} \subseteq A$ and $A \cap \{is\_same\_as(sofa_2, house)\} = \emptyset$). We express this using context as the learning from ASGs formalism does not currently support examples with specific inclusions or exclusions. Furthermore, since our generated ASG is guaranteed to be stratified, a parse tree has a unique answer set before the constraint in (6.1) is applied. Thus a single positive example with context in (6.1) is equivalent to a positive example with the context $\{is\_same\_as(sofa_2, sofa_1)\}$ and a negative example with the context $\{is\_same\_as(sofa_2, house)\}$.

Automated Labelling of Examples In order to automate the generation of the kind of rules in (6.1), we can simply reuse the work done in Subsection 6.1.2 for knowledge hunting. Specifically, let us consider again the $context$ function, which is defined as shown in Figure 6.2. Any nominal in the knowledge text which shares context with the schema's pronoun or one of its candidate referents is considered for labelling. We consider each match for the pronoun alongside each match for one of the candidates. Whether the label assigned is a positive one (i.e. an inclusion) or a negative one (an exclusion) depends on whether the two nominals have been marked as coreferents according to a deterministic process, which we describe in Section 7.2.1. We note that if one of the nominals is a pronoun and we are not sure of its referent, then that nominal will not be used to generate any context.

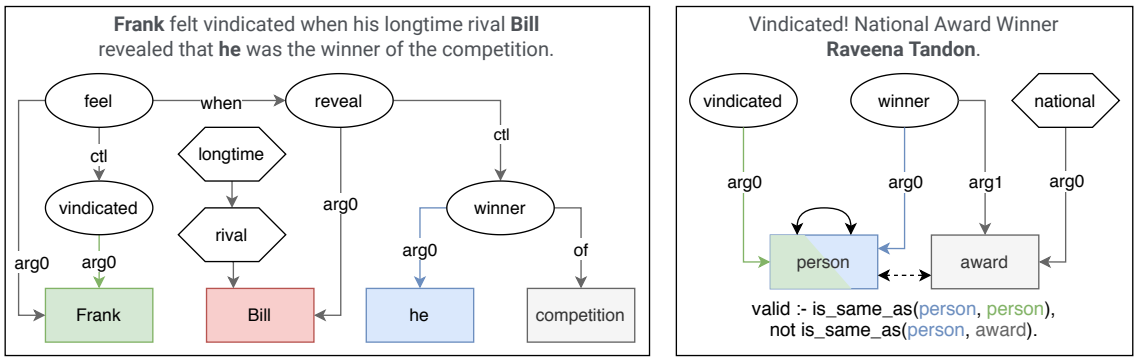
Figure 6.3 illustrates some examples of this process. In particular, we note that if there is no match for a candidate not linked to the pronoun (i.e. we are not able to extract a "negative example" from the knowledge text), then an exclusion is generated by choosing any other entity from the knowledge text (see Figure 6.3c). This is done in an attempt to avoid generating a learning task with no exclusions, which would result in a hypothesis that is too general. On the other hand, in the case where there is no match for a candidate linked to the pronoun (no "positive example", see Figure 6.3d), then we do not generate an inclusion. Adding an inclusion based on a nominal that does not share any context with a candidate from the schema makes



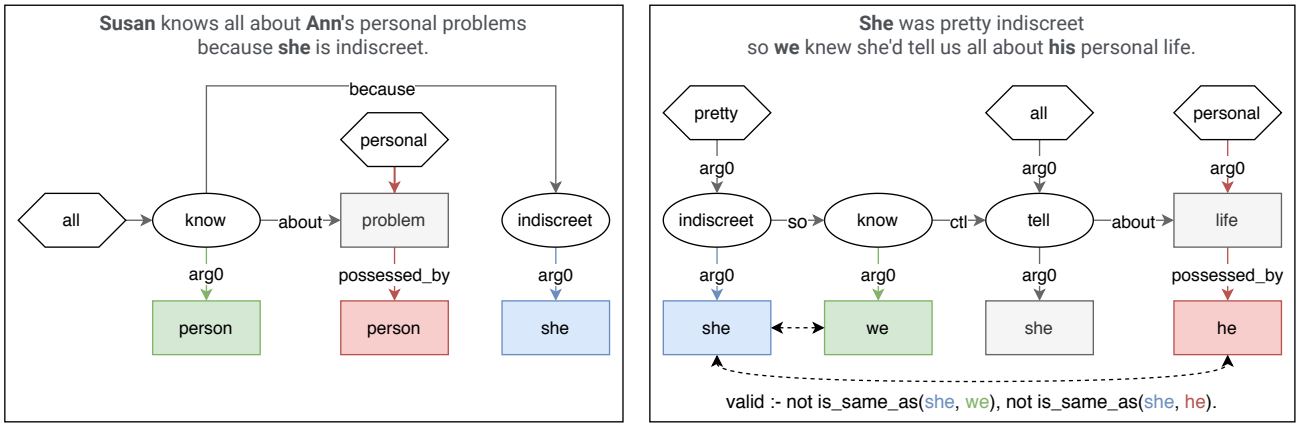
(a) Simple case based on our original example from Figure 6.2. The green and red/blue boxes show entities that have been matched to the candidates and pronoun respectively, based on shared context. The filled/dashed black lines are used to mark entities in the knowledge text which are known to be the same/different respectively.



(b) An example where one entity in the knowledge text is matched to both the pronoun and one of the candidates in the schema. This case can be handled in the same way as the simple case.



(c) An example where there is no nominal in the knowledge text which is matched to a candidate and is not matched to the pronoun. An exclusion is generated from some other nominal in the knowledge text to help rule out excessively general explanations.



(d) An example where there is no nominal in the knowledge text which is matched to both a candidate and a pronoun. No inclusion is generated, since it would require the learner to explain an additional concept, making the learning task significantly more challenging.

Figure 6.3: An overview of how we generate examples from knowledge texts (right), relative to Winograd schemas (left).

the learning task much more challenging but is never useful, as it only serves to force the learner to learn an additional concept that cannot be applied back to the schema. However, as we have mentioned, examples without any inclusions can still be useful as they force the learner to make its hypothesis more specific.

6.2.2 Hypothesis Space Generation

Given a set of examples with context describing valid/invalid interpretations, and a base ASG able to parse them, we now finally discuss how to automatically generate a hypothesis space for the learner to search through in order to find an explanation. In constructing a hypothesis space, there are four qualities of the final learning task that we must consider:

1. **Satisfiability.** The hypothesis space must be large enough to include rules capable of explaining the examples.

2. **Computational feasibility.** The hypothesis space must be small enough as to be generated and searched efficiently.

3. **Ability to handle noise.** Since we do not currently use weighted examples in our approach, optimising the size of the generated hypothesis space is key to ensuring that our approach handles noise well. Restricting the hypothesis space to a small number of simple explanations, in addition to making many tasks unsatisfiable, restricts the approach to shallow explanations. Meanwhile, increasing the hypothesis space not only significantly increases the computational cost of learning, but also allows us to "overfit" to the examples. A large hypothesis space allows us to propose very complex hypotheses which "encode the noise of our examples" by including highly specific conditions to handle these noisy cases.

4. **Quality of learned rules.** Different hypothesis spaces may result in rules being learned which result in the same final answer to a Winograd schema, but read very differently. For example, let us consider the schema "the fish ate the worm, it was tasty." Depending on the way in which the hypothesis space is constructed, examples of the form "I ate the tasty burger" might produce a rule of the form

   $$modifier(learned\_mod, tasty, X) \leftarrow event(\_, eat, \_, X).$$

   which reads as "if something is eaten, then it is tasty". At first, this may seem like an acceptable rule, but clearly it is not the case that *everything* that is eaten is tasty. Furthermore, let us note that the directionality of the implication in this case is artificial. As long as the examples talk about *eating* and something *being tasty* (which they are likely to do, given our knowledge hunting approach), we could have equivalently learned that "if something is tasty, then it is eaten". This is arguably a better rule in this case, but it is still technically incorrect. An alternative approach might be to produce constraints such as:

   $$\leftarrow event(\_, eat, \_, X), \text{not } modifier(\_, tasty, X).$$

   expressing "something is not eaten unless it is known to be tasty". But such a hypothesis clearly suffers from the exact same limitations. Overall, this kind of straight-forward

entailment does not occur in WSC problems. The rules learned rely on the context of the sentence. In it's most precise form, the knowledge needed to solve this schema is that "something that is eaten may be described as tasty, but something that eats may not", or more formally:

$$\forall XYZ.\,[eats(X,Y) \wedge tasty(Z) \wedge (X = Z \vee Y = Z) \rightarrow X = Z]$$

You might attempt to approximate these semantics with an ASP rule of the form:

$$\begin{aligned} is\_same\_as(X,Z) \leftarrow event(\_, eat, X, Y), modifier(\_, tasty, Z), \\ candidate(X,Z), candidate(Y,Z). \end{aligned} \tag{6.2}$$

where the predicate $candidate(X,Z)$ is defined appropriately, i.e. to mean X *may equal* Z. However, it is important to note that this formulation also suffers from several limitations — most notably, it results in longer rules with more variables (and so requires a larger hypothesis space).

Head Mode Declarations With the above considerations in mind, our hypothesis space aims to generate rules with $may\_refer\_to(P,C)$ at the head. The goal is to approximate the semantics of (6.2) by learning a rule of the form

$$may\_refer\_to(P,C) \leftarrow event(\_, eat, \_, C), modifier(\_, tasty, P).$$

which would then be combined with some background knowledge of the form:

$$is\_same\_as(P,C) \leftarrow may\_refer\_to(P,C), pron(P), cand(C).$$

This formulation has the benefit both of generating more relevant explanations (which do not read with an artificial implication), and also of simplifying the construction of the hypothesis space, since only one kind of head needs to be generated.

Automatic Generation of Body Mode Declarations In order to significantly constrain the hypothesis space while allowing for complex explanations, we generate mode body declarations based on the Winograd schema's knowledge graph. Specifically, hypotheses should be able to match on any subgraph of the schema's knowledge graph. Such an approach has the additional benefit that we can ensure that anything learned will be relevant — the learned rules will be applicable back to the schema itself.

The process is very straight-forward, combining a breadth-first-search with a simple translation. To demonstrate how it works, let us consider an example. We will once again consider the schema "the *trophy* doesn't fit in the brown *suitcase* because *it*'s too large." For reference, the knowledge graph for this schema is shown again in Figure 6.4. Firstly, let us note that the single head mode declaration uses the variables $pron$ and $cand$; specifically, it takes the form:

$$modeh(may\_refer\_to(var(pron), var(cand)))$$

We now traverse the knowledge graph from the pronoun and each of the candidates. We start at a depth of 0, adding just body declarations describing the initial nodes themselves, thus

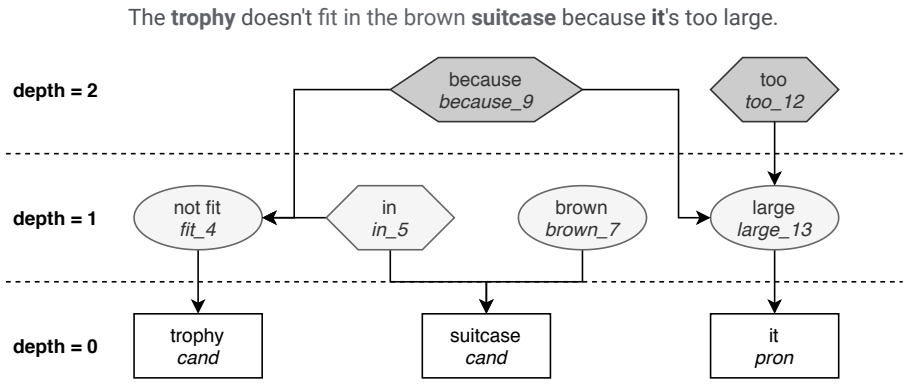


Figure 6.4: An overview of how we generate a hypothesis space from a knowledge graph. The process involves a breadth-first search from the candidate and pronoun nodes. Each node is translated into a body mode declaration. The variable type of each node is shown here in italics.

allowing the learner, for example, to specify that the candidate must be a trophy or must be a suitcase. Let $M_b(d)$ and $vars(d)$ denote the body mode declarations produced and variables seen at depth d, respectively. Then we have:

$$M_b(0) = \left\{ \begin{array}{c} modeb(nominal(var(cand), const(trophy))) \\ modeb(nominal(var(cand), const(suitcase))) \end{array} \right\}$$
$$vars(0) = \{pron, cand\}$$

The possible values for a constant type are generated based on its lemma and its sense, as well as any senses which it entails due to background knowledge. This may generate a very large number of possible values. For example, $suitcase$ generates the following possibilities:

$$constant(suitcase) = \{suitcase, bag\_n\_06, baggage\_n\_01, case\_n\_05, container\_n\_01\}$$

The process continues by following any edges from the variable seen so far, and adding these edges to the hypothesis space. At a depth of 1 we have:

$$M_b(1) = M_b(0) \cup \left\{ \begin{array}{c} modeb(event(var(large\_13), const(large), var(pron))) \\ modeb(neg\_event(var(fit\_4), const(fit), var(cand))) \\ modeb(modifier(var(into\_5), const(into), var(fit\_4), var(cand))) \\ modeb(event(var(brown\_7), const(brown), var(cand))) \end{array} \right\}$$
$$vars(1) = vars(0) \cup \{fit\_4, into\_5, brown\_7, large\_13\}$$

along with definitions for each of the constants, and at a depth of 2 we get:

$$M_b(2) = M_b(1) \cup \left\{ \begin{array}{c} modeb(modifier(var(because\_9), const(because), \\ var(fit\_4), var(large\_13))) \\ modeb(event(var(too\_12), const(too), var(large\_13))) \end{array} \right\}$$
$$vars(2) = vars(1) \cup \{because\_9, too\_12\}$$

at which point we have translated the entire graph, so we stop. Note that the variable types are based on IDs, to avoid any duplication — we want to constrain the hypothesis space as much as possible so that it can be generated and searched efficiently.

Learning Exceptions The main issue with our approach to generating mode declarations is that the learner does not have a lot of flexibility to explain edge cases. For example, consider again the schema "the *trophy* doesn't fit in the brown *suitcase* because *it*'s too large." Now suppose we are provided with the example "my *bills* won't fit in my *wallet* because *it*'s too small" (our knowledge hunting approach would never select such an example in practice, but we use it for demonstration, as its format is similar to some of the "tricky" examples that might be selected). This example would be labelled in such a way that the learner must make $is\_same\_as(it, wallet)$ true (since the wallet is the only singular noun in the sentence). The problem is that the learner does not have any way to make a hypothesis about something being too *small*, as the hypothesis space is generated from the schema alone, which only references the pronoun being too *large*.

One way to handle this case might be to also consider the examples when generating the hypothesis space. This would involve extending our approach above to consider all input knowledge graphs and merge variable types across the graphs where possible, and is an idea which has been explored in some depth by previous work (Lewis and Steedman, 2013; Chabierski et al., 2017), albeit in a slightly different context. However, the potential challenges of implementing such an approach are fourfold: firstly, it would significantly increase the size of the hypothesis space, especially as the number of examples are increased; secondly, it would allow the learner to easily encode the "noise" of noisy examples, rather than learning a more general rule which works *most of the time*; thirdly, it would mean the learner could now propose rules which may not be applicable back to the Winograd schema; and finally, it makes the learning task potentially quite challenging: instead of just having to learn the single concept relevant to the Winograd schema, the learner may now have to learn a variety of different concepts in order to fully explain the examples (e.g. the thing that is too large doesn't fit *and* the thing that is too small is not fit into). An alternative approach might be to allow negation as failure for body atoms, but this approach suffers from many of the same limitations.

Our approach instead allows many of these examples to be explained simply by means of learning *strong exceptions*[3]. This is done by adding the following head mode declaration:

$$modeh(cannot\_refer\_to(var(pron), var(cand)))$$

This allows the conclusion that some P *may refer to* some C to be refuted. Since ILASP does not support strong negation, these semantics are achieved simply by extending the background knowledge:

$$is\_same\_as(P, C) \leftarrow may\_refer\_to(P, C), \text{not } cannot\_refer\_to(P, C), pron(P), cand(C).$$

In particular, we note how our tricky example can now explained by learning:

$$may\_refer\_to(P, C) \leftarrow pron(P), neg\_event(F, fit, \_), modifier(\_, into, F, C).$$
$$cannot\_refer\_to(P, C) \leftarrow modifier(\_, large, P), neg\_event(F, fit, \_), modifier(\_, into, F, C).$$

[3]One other avenue which might be worth exploring in future work is the use of *weighted examples*, which might be able to handle these exceptional cases as, in our experience, they are relatively rare. The weights could also encode our confidence of the relevance of different examples, allowing the least relevant examples to be mislabelled at a smaller cost than the most relevant ones. However, compared to weighted examples, allowing the learning of exceptions provides the additional benefit that it is sometimes actually desirable to explain a concept in terms of an exception, as it may produce a simpler hypothesis than some direct but complex argument.

I.e. we have learned an exception, that the thing that is large *does not refer to* the thing which is not fit into.

Optimising the Size of the Hypothesis Space The approach we have discussed thus far generates hypothesis spaces which are far too large to search, or indeed even generate. The most significant issue arises from the use of constants which can take many values. Each set of constants C causes the hypothesis space to increase in size by roughly $|C|$ times, which is a clear issue for our approach given the number of constants we generate. We propose four extensions to our approach to ensure that the hypothesis space does not grow too large:

1. **Pruning based on examples.** The first and most significant extension is that we reduce the search space by removing any atoms which do not occur in any of the examples. Most notably, a potential value for a constant is discarded if it never appears in any of the examples. If the set of values for a constant becomes empty, then any body mode declaration using that constant may also be discarded. Additionally, if there is some overlap in the set of senses produced by a word in the schema and a word in the example, then we can retain only their lowest common ancestor. Say, for example, that the only word from an example which generates a sense in $constant(suitcase)$ is the word *briefcase*. Then all of the values for the $suitcase$ constant, except for $case\_n\_05$ which is the lowest ancestor of the two words, could be discarded.

2. **Limited search depth.** The reason for our breadth-first-search approach is that it allows us to easily restrict the search to a given depth in order to keep the size of the hypothesis space manageable. This is particularly useful in longer Winograd schemas, which can produce very large and deep knowledge graphs. In particular, we currently limit the depth of the search to 3, as we find that explanations rarely require context that is more than three nodes away from the pronoun or candidates. Even if they do, the necessary explanation is usually then too complex for us to generate in an acceptable amount of time.

3. **No negation as failure.** We currently enforce that all atoms in learned rules are positive. Although negation as failure does allow us to correctly answer a small number of schemas that cannot be answered without it, we find that explanations which use negation as failure are usually very difficult to interpret (e.g. *the person who speaks is the person who breaks their something that isn't concentration*). Furthermore, excluding negation as failure from the hypothesis space significantly reduces the time to generate the search space.

4. **Mode bias constraints.** Finally, we define a set of generic constraints over the hypothesis space to rule out a significant number of hypotheses that we can determine, from their structure alone, to be unsuitable. In total there are eighteen such rules and constraints, taking three forms.

   Firstly, we use the constraint

   $$\leftarrow body(holds\_at\_node(nominal(anon(\_),\_),\_)).$$

to prevent any hypothesis which includes a nominal without linking that nominal to some event or modifier. A rule which breaks this constraint would imply that the mere presence of some nominal is enough to influence the result, which we deem to be extremely unlikely.

The second and third set of rules are used to similarly exclude modifiers and events which are not linked in any way to the argument structure. For example, consider the rules:

$$has\_context(X) \leftarrow body(holds\_at\_node(modifier(\_,\_,X,\_),\_)).$$
$$has\_context(X) \leftarrow body(holds\_at\_node(modifier(\_,\_,\_,X),\_)).$$
$$\dots$$
$$\leftarrow body(holds\_at\_node(event(X,\_,anon(\_),anon(\_)),\_)), \text{not } has\_context(X).$$
$$\dots$$

The first two rule express that an event has context if there is a body atom that modifies it; the constraint then asserts that an event must either specify at least one of its arguments, or be described by some context.

Other Constraints Some final constraints on our hypothesis space, which ILASP requires us to set, are chosen as follows:

- **Maximum rule length and maximum number of literals.** The number of body literals permitted per rule is capped at 5 and the rule length at 6 (increased from their defaults of 3 and 5 respectively). This allows the pronoun and candidate to simultaneously be specified by two or three atoms each, thus fairly complex rules can be learned. There are a small number of examples that may be explained by allowing longer rules, however we determined the performance cost of increasing the limit to be too great[4].
- **Maximum number of variables.** Initially, an upper bound on the number of variables was established based on the number of variable types in the mode declarations. The number of variables was later capped at 12 in order to improve performance, as we never observed more than 12 variables being used (although there is one schema which uses exactly 12 variables).
- **Recall.** For body atoms, the recall is always set to 1, as our hypothesis space is structured in such a way that one body mode declaration should not need to be used multiple times. We do not limit the recall for head mode declarations, i.e. multiple rules may be learned.
- **Constraints and choice rules.** We do not allow constraints or choice rules to be learned.

[4]Increasing the maximum rule length to 7 increases the average size of the hypothesis space by over 100 times — to approximately 40 000 rules.

6.3 Applying Learned Knowledge to Winograd Schemas

Each learning task generated by the method we have described in Section 6.2 produces as a result, either an ASG extended with commonsense rules that explain the provided examples, or an assertion that the learning task is unsatisfiable (i.e. there is no hypothesis in our hypothesis space capable of explaining the examples). The process for applying the learnings back to the original schema is then an extremely straight-forward process.

In particular, let us consider again our running example of "the *trophy* doesn't fit in the brown *suitcase* because *it*'s too large." Given suitably selected examples, the returned ASG will include the learned rule:

```
start -> s {
  may_refer_to(P, C) :- event(_, large, P), neg_event(_, fit, C).
  ...
}
```

Hence if the semantic parse of a sentence introduces an object which *is large* and another object which *doesn't fit* at the root node, we determine that the thing that *is large* may refer to the thing that *doesn't fit*. This rule is combined with background knowledge in a similar way as in the learning tasks themselves:

```
#background {
  is_same_as(P, C) :- pron(P), cand(C), may_refer_to(P, C),
                      not cannot_refer_to(P, C).
  ...
}
```

The Winograd schemas are passed to the ASG with context to specify their pronoun p ($pron(p)$) and two candidates $c1$ and $c2$ ($cand(c1)$ and $cand(c2)$). Then the candidate $c1$ will be returned as the answer if $is\_same\_as(p, c1)$ holds at the `start` node *and* $is\_same\_as(p, c2)$ does not[5]. This can be checked by ensuring that when the former literal is provided as context, the generated program is satisfiable, and when the latter is provided, it becomes unsatisfiable. Checking whether $c2$ is the answer is handled in the same way.

Generating Natural Language Explanations To generate natural language explanations, we extract the learned rules from the returned ASG, and translate those rules automatically into English by means of a set of simple templates. Different templates exist for different ways in which the two concepts are connected in the hypothesis. For example if the candidate is connected to the pronoun in the hypothesis by means of a modifier with the lemma *because*,

[5]Note that if the examples selected provide support for each candidate and do not rule out either of them, then it's perfectly acceptable for the hypothesis to make both of these literals hold, in which case no result should be returned.

then a template of the form "... *causes* ..." would be matched. As a concrete example, let us consider the rule:

$$may\_refer\_to(P, C) \leftarrow event(\_, large, P), neg\_event(\_, fit, C).$$

This simple form of rule, in which the pronoun and candidate are not connected, matches the pattern:

The {*variable*} {*wh-word*} {*event1*} {*be*} the {*variable*} {*wh-word*} {*event2*}.

The word for the *variable* is selected from *person*, *people*, *thing* or *things* based on the number and animacy of the candidates in the schema. In this case, the trophy and the suitcase are both singular and inanimate, so the word *thing* would be chosen. The *wh-word* is selected similarly from *who* and *which*, and the verb to-*be* is conjugated appropriately.

The strings for *events* are generated based on (a) which argument the variable is placed in, (b) whether the event corresponds an adjectival or verbal lemma, (c) whether the verb is negated or not, (d) any other arguments which have been specified and (e) any prepositions or adverbial modifiers to the event. We use existing tools to conjugate the verb accordingly. In this case *event1* would generate the string "doesn't fit" and *event2* would generate the string "is large", giving the final explanation:

The thing which doesn't fit is the thing which is large.

Chapter 7

Implementation

7.1 Base ASG Generation

The main implementation challenge in this project is in developing a system capable of automatically generating base ASGs. Our system depends on several large machine learning models and libraries, and requires a significant amount of logic to process and combine the results from these resources into a meaningful form, from which a base ASG can be generated.

7.1.1 Design

Figure 7.1 shows an overview of the internal design of our system. To improve code organisation, our implementation is divided into three packages. The first parses an input text, the second annotates a parse tree with semantic information which is used during ASG generation, and the third generates and prints the base ASG itself from a set of given parse trees. We will now briefly describe each of the main components.

Parsing The parsing package (shown in red in Figure 7.1) combines the results from several machine learning models into a `HPSGTree` structure. Each non-leaf node in the tree stores its constituent type, dependency label, children, and the index of its head node. Leaf nodes additionally store a pointer to their respective `HPSGToken`. The token structure stores the token ID, original value, part-of-speech, lemma, NER class, WordNet sense, and possible referents.

The tokeniser acts mainly as an adaptor to the preprocessing library, allowing it easily to be swapped out for any other library capable of providing the same functionality, such as spaCy, NLTK or Stanza. It also makes a call to a separate WSD model, which has its own lightweight adaptor. The parser similarly acts mainly as an adaptor for constituency and dependency parsers.

The final module of the parser finds and replaces certain patterns in the tree. These are sparingly used fixes for known parser bugs. For example, the model we use for dependency parsing appears to be unfamiliar with *so* as a coordinating conjunction and consistently leaves such coordination unlabelled. We therefore have rules to find such unlabelled dependencies and make a best-effort guess about what the label ought to be.

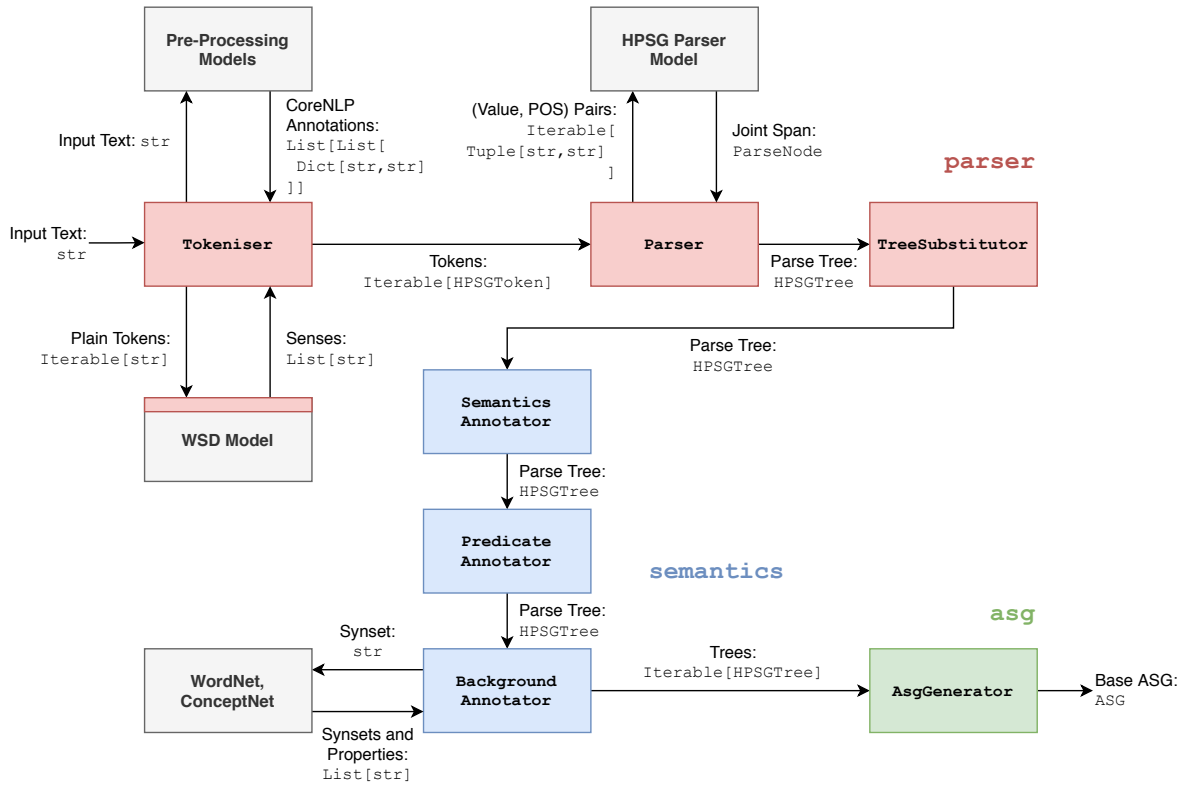


Figure 7.1: Pipeline for generating base answer set grammars.

Semantics The semantics package (shown in blue in Figure 7.1) is where the bulk of processing is done. Each module here is an annotator, implemented using a visitor pattern, which simply takes a `HPSGTree`, adds some additional information, and returns the updated tree. These annotators are then chained together.

The semantics annotator is responsible for determining the head and dependents at each node, checking their types, and noting the relevant transformation. The predicate annotator then takes these annotations and works out the exact logical forms at each node, and how the annotations should be combined (see Subsection 5.3). Finally, the background annotator determines entailed senses from WordNet and properties from ConceptNet (see Subsection 5.3.2).

ASG Generation The ASG package (shown in green in Figure 7.1) takes a set of input trees, and produces an ASG to encode their syntactic and semantic composition. The ASG representation is a map from production rules to sets of ASP rules, each of which is represented by its own structured `AspRule` datatype. This allows for rules to be easily modified, combined and extended. Producing the base ASG involves a final walk over each parse tree, to generate the actual ASP rules (now mostly a straight-forward process) and save them according to their production rules. It also involves generating a rule for the `start` node. Our implementation supports two kinds of ASGs: grounded (which use the "lexicalised" grammar rules discussed in Subsection 5.3) and ungrounded ("non-lexicalised").

7.1.2 NLP Dependencies

We now briefly discuss and motivate our choice of the libraries and models which our approach depends on.

Preprocessing We use CoreNLP (Manning et al., 2014) for sentence splitting, tokenisation, lemmatisation, part-of-speech tagging and named entity recognition. We briefly discuss the relevance of these tasks and how they are implemented in Subsection 4.1.1. The motivation for using CoreNLP is that it allows us to run a single server in the background which is able to do the vast majority of preprocessing tasks that we require. Furthermore, many of the models achieve near state-of-the-art performance, which is important since incorrect POS tags severely handicap the parser, and correct lemmatisation and NER is vital for being able to match entities of the same type.

We also use CoreNLP for *deterministic* coreference resolution. In this project, we are only ever interested in annotating references between entities that we can be *sure* are the same — it is our job to resolve the ambiguous cases! State-of-the-art approaches to coreference resolution use neural models that are rarely confident in their predictions. As such, we instead rely on a much more traditional sieve-based approach. In this approach, a list of possible referents is generated, and passed through several "sieves", each of which discards impossible referents, for example, based on gender, number and animacy. In a traditional setting, the remaining referents would then be ranked based on linguistic features and the top result selected, however we instead select a referent only if there is just one to choose from out of the final sieve.

Word Sense Disambiguation WSD is a particularly challenging problem, and it is only recently that models have begun to achieve accuracies which significantly improve on just choosing the most common sense. Our main method of handling this difficulty is to rely very little on WSD: it is used solely for generating background knowledge and so when it goes wrong it's unlikely to have a significant impact on downstream learning tasks. However, it is of course helpful to determine senses accurately to improve the quality of background knowledge. As such, we use the open-source disambiguate model (Vial et al., 2019), based on RoBERTa, which uses sense vocabulary compression in order to achieve near state-of-the-art performance.

Parsing For constituency and dependency parsing, we use the HPSG neural parsing approach developed by Zhou and Zhao (2019). It uses an encoder-decoder architecture to predict such a joint span structure from a tokenized sentence. In particular, each token's representation consists of character, word and part-of-speech embeddings. The parsing model is inspired by (Kitaev and Klein, 2018): the encoder consists of twelve self-attention layers, and summarizes an input sentence into a set of vectors; meanwhile the decoder is trained to assign scores to each possible parse tree, based on these vector summaries. At test time, the parse tree with the highest score is selected.

Our motivation for using this approach is twofold. Firstly, the model is trained with a joint training goal of predicting both constituent and dependency structures, and thus generally

produces realistic structures where the two align, making our task of generating an ASG from the structure far simpler. Secondly, it's a modern approach, based on XLNet embeddings, which achieves near state-of-the-art performance.

7.2 Learning

7.2.1 Design

Figure 7.2 shows an overview of the internal design of our system for learning commonsense knowledge for the WSC using answer set grammars. The components are organised into three packages: one to convert examples from different datasets into a consistent form, a second to handle the knowledge hunting process and a final one to handle the generation and solving of learning tasks. We will now briefly discuss some of the implementation choices for each of these packages.

Datasets We have an internal `Schema` type which is used for all Winograd schemas and examples, and encodes the source of those examples, the pre-processed text, whether the example has its coreference labelled, and if so, what the pronoun and candidates are. The dataset package (shown in yellow in Figure 7.2) consists of several simple components to translate examples from different sources into this standardised format.

Knowledge Hunting The knowledge hunting package (shown in orange in Figure 7.2) is the largest and most complex of the three packages.

The two labellers are responsible for converting `Schemas` into a much more useful format. The `Example` datatype contains the ASG generated from the sentence, which is generated using the system described in Section 7.1. It also contains an internal representation of the text's knowledge graph. The labellers are also responsible for marking the pronoun and candidates in the knowledge graph and generating the set of patterns that correspond to the *context* function described in Subsection 6.1.2. Most of the code is shared between the two labellers, the only difference being how the pronoun and candidates are identified:

- In the case of Winograd schemas, we currently use a very crude approach which attempts to match the exact text for the candidates against a nominal in the knowledge graph. This actually turns out to be a significant limitation of our system: for example, the Winograd schema "*Joe* has sold *his house* and bought *a new one* a few miles away; he will be moving out of *it* on Thursday" describes its two candidates as "the old house" and "the new house", which do not match any nominals in the semantic graph and hence this schema is left unlabelled and unattempted. In most cases, this problem does not look particularly difficult to solve: even a straight-forward approach of comparing the word-embeddings of each of the candidates with each of the nominals in the semantic graph and choosing the assignment which maximises the sum of their similarities would likely be sufficient.

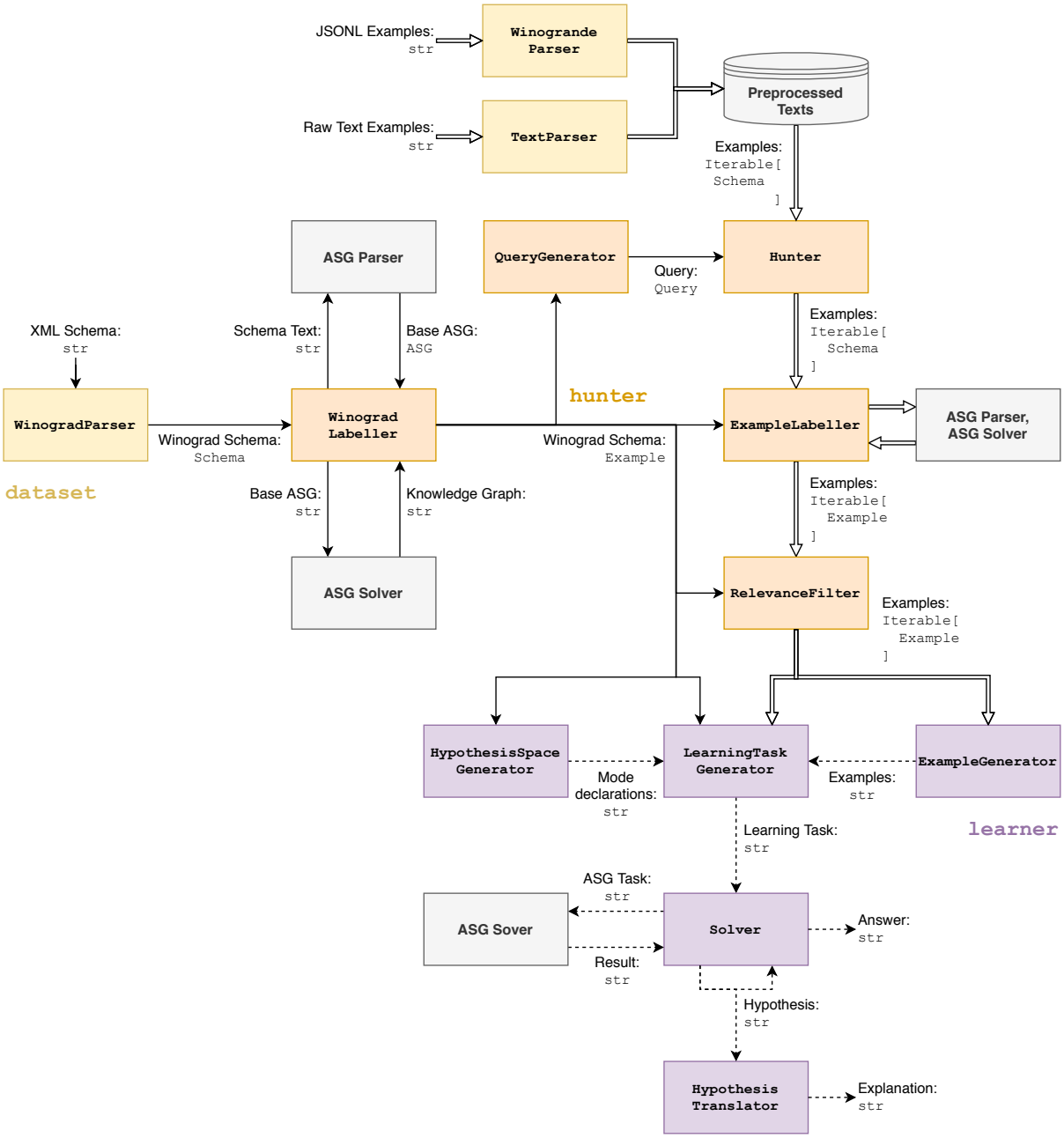


Figure 7.2: Pipeline for learning commonsense knowledge for the WSC. Single lines are used to show the path of the Winograd schema through the pipeline, and double lines to show the path of examples.

- In the case of examples, the labelling is done by matching context with the schema. It is important to note that this is done even for examples from sources with existing labels. For example, consider the example "back when I was obese, I ate too many *snacks* and ate too few *vegetables* but my weight reacted best when the *vegetables* were tasty," which comes from the WinoGrande dataset. The existing labels are not ideal for learning knowledge for the schema "*the fish* ate *the worm*; *it* was tasty", since the negative label, *snacks*, is used to express a very different concept to the one we want to learn (namely, that somebody loses weight best when healthy food is tasty). However, our context-based labelling approach finds that the knowledge we want to learn (the thing that is eaten is tasty, not the thing that eats) is also expressed unambiguously in this example, and thus chooses the much more useful negative label, *I*.

The query generator component takes a schema and generates a query to run over the full set of examples, based on the approach described in Subsection 6.1.1, while the hunter actually compiles and runs the query. Examples are labelled only after the initial query as the labelling process is computationally expensive. The following and final component of the knowledge hunter is the relevance filter, which ranks each example with respect to the schema, according to the metric proposed in Subsection 6.1.2, and returns the top-N examples.

Learning Finally, the learning package (shown in purple in Figure 7.2) implements the approach described in Section 6.2. In particular, the learning task generator combines the ASG rules for the schema and all of the examples, the mode declarations produced from the schema, the example definitions produced from the knowledge texts, and a fixed background knowledge into a complete learning task. This learning task is then passed to the solver, which acts mostly as an adaptor to an existing library for learning and solving ASGs.

7.2.2 Knowledge Sources

We provide a brief overview of the knowledge sources considered for our knowledge hunting process, and their qualities, in Table 7.1. The desirable properties of a knowledge source are that it should contain a large number of examples, including many examples from similar domains as to the WSC, and that those examples should have unambiguous (preferably hand-labelled) coreferences. Additionally, we strongly prefer sources that allow us to both pre-process these examples and automate our search process over them.

Previous approaches to knowledge hunting rely on web search queries (Sharma et al., 2015; Emami et al., 2018; Prakash et al., 2019; Sharma, 2019), which has the benefit that it allows a huge amount of text to be searched efficiently with relatively little work by the implementer. However, automating web searches introduces legal issues as it technically breaks the terms-of-service of almost all major search providers, and furthermore, the search operators offered by these providers are rather limited — there is no way, for example to search for specific lemmas, senses or parts of speech, and so searching for similar sentences while allowing different person, number, tense etc. is a difficult problem that requires an excessive number of individual search terms.

Knowledge Source	Size	Similar domain	Labelled references	Search can be automated	Can preprocess before search
Web Search Engine	Huge	✗	✗	✗	✗
Web Corpus (e.g. CommonCrawl)	Large	✗	✗	✓	✓
Related Corpus (e.g. STORIES)	Medium	•	✗	✓	✓
WinoGrande/DPR	Small	•	✓	✓	✓
Previous Knowledge Hunting Results	Tiny	✓	✗	✓	✓
Handwritten	Tiny	✓	✓or ✗	✓	✓

Table 7.1: Qualities of knowledge sources considered.

Our approach relies on three sources of knowledge. The first is WinoGrande, which provides some 40 000 labelled WSC-like examples. We find it to be particularly useful as there is a fair amount of overlap between the commonsense knowledge required to answer WinoGrande and that required to answer WSC. Furthermore, the size of the WinoGrande means that it is possible to pre-process the entire dataset in a number of hours, and search it in an number of milliseconds on consumer hardware. The second source is made up from the results of previous knowledge hunting approaches to the WSC, which are sourced from web search queries. These include several hundred sentences which are highly similar to Winograd schemas, but they are not labelled and are not guaranteed to have unambiguous coreferences or express the correct knowledge required to solve the schemas (and of course there is no guarantee that our own knowledge hunting approach will select them as examples). Finally, we allow for a small number of handwritten examples to be included. It is important to note that, unlike some other approaches (Sharma, 2019), we *do not use* handwritten or hand-selected examples when evaluating the accuracy of our approach, but we do use them to demonstrate the capability of our approach to generate high-quality explanations based on the examples which have been observed.

7.2.3 Dependencies

The learning process has only three significant dependencies.

CoreNLP The learner of course relies heavily on our base ASG generator. In particular, the results of pre-processing using CoreNLP are used in several places. The deterministic (sieve-based) coreference resolver is used to ensure we only label nominals whose identity we are

sure of. Furthermore, the number and animacy of candidates is used to generate natural language explanations as described Section 6.3.

ASG Solver (and `ILASP` and `clingo`) The learner relies heavily on an existing program for learning and solving ASGs (Law et al., 2019). This program takes a learning task and returns whether the task is satisfiable, and the optimal hypothesis if it is. We make only very minor changes to its source code, such as adding an option to print the ASP meta-representation (so that it can be passed to an ASP solver such as `clingo`, allowing a full knowledge graph to be extracted), disabling some features which we do not use in order to improve performance (e.g. the ability to learn annotated atoms), and adding some checkpoints in order to identify performance issues.

Pattern We use the web-mining library, `pattern` (Smedt and Daelemans, 2012), to conjugate verbs when generating natural language explanations, as described in Section 6.3.

Chapter 8

Evaluation

8.1 Knowledge Graphs and Base ASG Generation

In this section, we will evaluate both the quality of our knowledge graph representation and our automated approach for generating these structures using answer set grammars. We will first evaluate the claim that our generated ASGs can act as semantic parsers, by exploring their ability to produce accurate meaning representations for Winograd schemas. We will then critically analyse the quality of our semantic parses, by comparing the semantic richness of our representation against the representations used by several freely-available and widely-used broad-coverage semantic parsers.

8.1.1 Performance on WSC

In our first experiment, we generate a base ASG for each Winograd schema. We then parse each schema with its respective ASG, and extract the knowledge graph structure produced at the root node. This produces a set of atoms of the form:

$$
\begin{gathered}
event(yelled, yell, jim) \\
\wedge event(upset, upset, he) \\
\wedge nominal(jim, person) \\
\wedge nominal(kevin, person) \\
\wedge nominal(he, he) \\
\wedge modifier(at, at, yelled, kevin) \\
\wedge modifier(because, because, yelled, upset) \\
\wedge modifier(so, so, upset)
\end{gathered}
$$

We choose to focus our evaluation on the ability of the procedure to extract the (core and non-core) *argument structure* of the text. Therefore, we then apply a post-processing step to separate out each argument, creating a (*source*, *label*, *argument*) triple, similar to in a neo-Davidsonian representation. The core arguments of events are assigned the labels *agent*, *patient*, *recipient* and *controls* respectively. Non-core arguments (modifiers) use their own value

as their label. For example, the events and modifiers in the conjunction above generate the triples:

$$(yelled, agent, jim)$$
$$(upset, agent, he)$$
$$(yelled, at, kevin)$$
$$(yelled, because, upset)$$
$$(\_, so, upset)$$

Note how we do not evaluate the lemmas/NER categories assigned to each node in the knowledge graph, instead using the token values directly. Evaluating these "node labels" would not be worthwhile since CoreNLP's lemmatisation and basic 3-class NER classifier almost never produce errors. We also choose not to evaluate senses assigned to each token during the background knowledge generation process: while this process commonly produces errors, these errors can be nuanced and difficult to spot. They also do not affect our base representation and are rarely harmful in practice, since we can rely solely on lemmas if required.

We next defined gold representations for each of the 273 Winograd schemas, by manually identifying the core and non-core arguments in each text and labelling them according to the format described above. We then measure labelled accuracy; that is we consider each triple as a whole: a predicted edge is only counted as a true positive if the exact edge is present in the gold representation. Using this definition, we find 76 false positives (generated edges not present in the gold representation) and 88 false negatives (gold edges not generated), giving the approach an overall precision of **0.972** and a recall of **0.968**. The recall is slightly worse than the precision as there are a couple of cases where missing dependencies cause us to lose an entire sub-phrase in our generated representation.

We note that the precision and recall of our approach is in the region of what is achieved by a dependency parse alone. This substantiates our hypothesis that our approach does not introduce many more inaccuracies beyond those which arise from the machine learning models used for pre-processing and parsing. This is largely a result of a key design choice of our approach, which is that we would only use semantic information which we could determine accurately. In particular, we only have four kinds of core argument, which can largely be determined from the dependency structure, and non-core arguments use lemmas directly from the text. This is in contrast to, say, FrameNet style argument structure which has five kinds of core argument and nineteen distinct classes of adjunctive arguments: classifying arguments in such a structure is clearly much more difficult and will produce many more errors.

Investigating more deeply, we find that almost all of the errors in our generated representations result from inaccurate dependency parses. The only exceptions are one case in which the minimal distance policy fails and a handful of examples of raising (see Subsection 5.3.1), as well as few cases where a passive construction has no auxiliary verb, and as such we fail to detect that the verb is passive. There are even some cases where the POS-tagging or dependency structure is incorrect, however we are still able to generate the correct semantic representation since we detect heads/dependents of the incorrect type and correct them automatically.

In Table 8.1, we briefly compare our approach with several other openly available semantic parsers based on a variety of different formalisms. Unfortunately, directly comparing the ac-

curacy of broad-coverage semantic parsers is a difficult task due to the variety of different semantic representations. Regardless, Table 8.1 makes very clear the trade-off between accuracy and semantic richness. For example, AMR relies on PropBank framesets and significantly abstracts away from syntactic features. This means that it is able to unify structures that our approach cannot. However, it also makes the semantic parsing task significantly more difficult, as demonstrated by the fact that in more than two-thirds of cases, the AMREager parser fails to extract even the general argument structure of the schema. We will investigate the properties shown in this table and the overall quality of our semantic representation more thoroughly in the next section.

8.1.2 Quality of Knowledge Graphs

We will now look in more detail at some of the strengths and limitations of the knowledge graph structures we generate and our method for generating them.

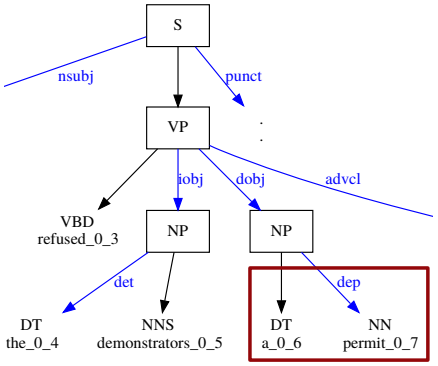
8.1.2.1 Strengths

Accuracy The graphs generated by our approach are generally highly accurate: they are based on constituency and dependency structures which can often be determined accurately, and as demonstrated in the previous subsection, we use a translation approach which is typed and linguistically-motivated, and as such very rarely introduces additional errors. Indeed, we even occasionally fix existing parsing errors. Figure 8.1 shows an example, where the determiner has incorrectly been marked as the head of a noun phrase. We are able to resolve this and choose the noun since we have a type constraint stating that the direct object of an event must be a nominal.

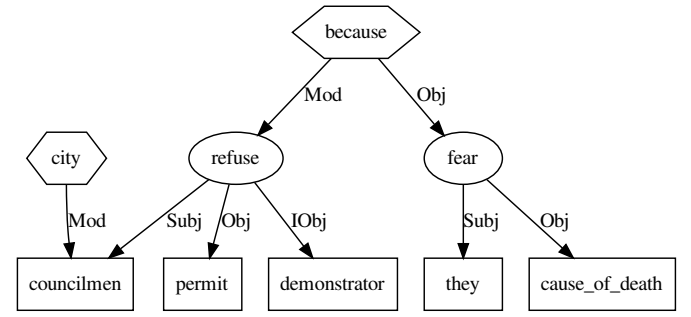
Overall, we parse 219 (80%) of schemas perfectly and generate the correct general argument structure for 267 (98%) schemas. More ambitious approaches, such as those which produce AMR, can fail on even very simple sentences, as shown in Figure 8.2.

Evaluating the accuracy of semantic parsers on the WSC is time consuming, so we only include the results of our own approach and AMREager in Table 8.1. However, Sharma (2019) notes that "40 [WSC] problems [could] not [be] answered because of syntactic dependency parsing errors and part-of-speech errors while generating the representations", suggesting that at most 233 problems are parsed correctly by their approach, and we also found that at least 90 clearly incorrect parses were generated by OpenCCG.

Robustness The approach to generating graphs is robust, and supports both complex linguistic structures, as well as ungrammatical texts. Table 8.2 lists the set of linguistic constructions which we can parse without failure, while Figure 8.3 demonstrates our ability to generate a knowledge graph for the ungrammatical text "PSU too large, doesn't fit". As noted by Reddy (2017), this is a key advantage of dependency-based approaches to semantic parses compared to CCG parsers, which generally must satisfy the constraint that the text is eventually assigned a sentence tag. This makes our approach suitable for parsing text sourced from the web, where ungrammatical sentences occur frequently, for example in news article titles or forum posts.

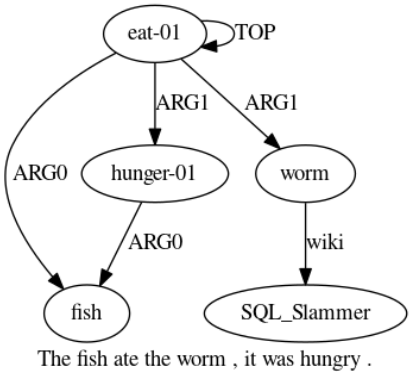


(a) Section of the parse tree demonstrating incorrect dependency structure.

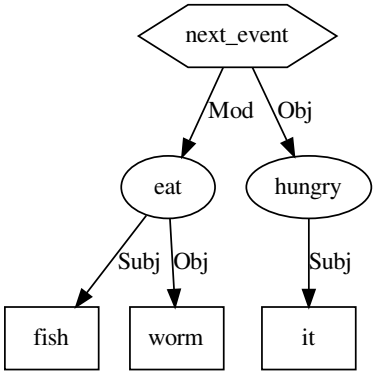


(b) Generated knowledge graph.

Figure 8.1: Demonstration of robustness to parsing errors.



(a) AMREager representation.



(b) Our representation.

Figure 8.2: Semantic Representations for the simple sentence “The fish ate the worm, it was hungry.”

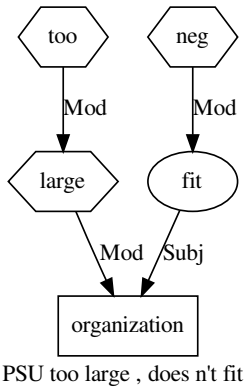


Figure 8.3: Demonstration of robustness to ungrammatical texts.

Property	**Dependencies** (Zhou and Zhao, 2019)	**DepLambda** (Reddy, 2017)	**OpenCCG** (White, 2016)	**Chabierski** (Chabierski et al., 2017)	**Wino-Grammar** (Ours)	**TRIPS** (Allen and Teng, 2017)	**KParser** (Sharma, 2019)	**AMREager** (Damonte et al., 2017)
Base formalism	Dependencies	Dependencies	CCG	CCG	ASG (Simplified HPSG)	Construction Grammar	Dependencies	None (Predicts directly from tokens)
Accuracy (WSC273 schemas parsed perfectly)	221 (81.0%)	-	-	-	219 (80.1%)	-	-	43 (15.8%)
Accuracy (WSC273 schemas parsed with correct arg structure)	-	-	-	-	267 (97.8%)	-	-	89 (32.6%)
Semantic richness	Poor	Mediocre	Good	Good	Good	Very Good	Very Good	**Excellent**
Domain general approach?	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Optimised representation for inductive learning?	No	No	No	**Yes**	**Yes**	No	No	No

Table 8.1: A comparison of semantic parsers based on their suitability for inductive learning approaches on the WSC. It is important to note that the accuracy results here are not directly comparable since each parser in this table uses a different semantic representation. However, we note that richer semantic representations are more difficult to generate and thus usually correspond with lower accuracy.

Property		Dependencies	DepLambda	OpenCCG	Chabierski	WinoGrammar	TRIPS	KParser	AMREager
Key representation features	*Semantic roles*	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
	Preposition senses	No	No	No	No	Limited	**Yes**	**Yes**	**Yes**
	Discourse relations	No	No	No	No	Limited	**Yes**	**Yes**	No
	Cross-POS relations	No	No	No	No	No	No	No	**Yes**
Lexical properties	*Semantic mapping to WordNet (or similar)*	No	No	**Yes**	No	**Yes**	**Yes**	**Yes**	**Yes**
	Entailment	No	No	No	No	**Yes**	No	**Yes *****	No
	Named entities	No	No	**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**
	Compound nouns	**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**	**Yes**	**Yes**
	Tense	No	No	**Yes**	No	No	**Yes**	**Yes**	No
Supported linguistic structures	*Entity attributes (including copular form)*	No	**Yes**	**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**
	Possession	**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**	**Yes**	**Yes**
	Passives	No	No	No	**Yes**	**Yes**	No	**Yes**	**Yes**
	Relative Clauses	No	**Yes ****	**Yes**	**Yes**	**Yes**	**Yes**	No	**Yes**
	Control and raising	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	No	**Yes**
	Coreferencing	No	No	No	**Yes**	**Yes ***	No	**Yes**	**Yes**
Supported logical structures	*Negation*	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
	Conjunction	No	**Yes ****	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
	Quantification	No	No	No	**Yes**	No	**Yes**	No	No

*Unambiguous coreferences only. **Requires post-processing steps. ***Hypernymy only.

Table 8.2: Overview of the properties represented by our knowledge graphs and constructions supported by our automated approach, compared to previous work on broad-coverage semantic parsing.

Canonical Representations Our approach significantly improves upon dependency parsing alone, by assigning many structures with different dependency structures the same knowledge graph. In particular, as shown in Table 8.2, we handle copular form, passives and relative clauses without the need for specialised predicates, in addition to sharing arguments in constructions such as control, raising and coordination. Some examples of different constructions receiving the same representation are shown in Figure 8.4, and further examples are given in Subsection 5.3.1. As shown in Table 8.2, the only other representations we looked at which treat all of these constructions in a uniform way are the representation used by Chabierski et al. (2017) and AMR.

Consistency One benefit of the use of answer set grammars results is that knowledge graphs are generated in a systematic way, that is easy to follow and verify. Another result of this is that the same linguistic structures consistently generate the same patterns in knowledge graphs. Meanwhile, machine-learning based approaches which predict semantic structure directly from a sequence of tokens, may generate entirely different structures for sentences which are identical, save a single noun for example.

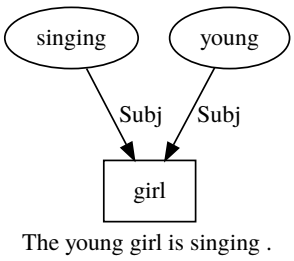
Suitability for ILP As discussed in Section 5.1, we deliberately choose to implement a classical Davidsonian semantics instead of a more popular neo-Davidosnian semantics. The reasoning for this is that, despite increasing the complexity of generating representations, it leads to shorter representations that can be described with fewer variables, which is a useful quality for reducing the size of the hypothesis space for ILP techniques. Additionally, and for the same reason, we use a very small number of fixed predicates (just three), in contrast to most other approaches which have a different predicate for every different concept. As pointed out in Table 8.1, the only other approach we have found which makes these considerations is the one proposed by Chabierski et al. (2017).

Modularity As noted in Section 7.1, we have dependencies on a preprocessor (tokeniser, lemmatiser and basic NER classifier), a constituency parser and a dependency parser. Each of these components just needs to implement a simple interface, allowing us to easily swap out one model for another. Thus we can improve the performance of our system at very little cost as the state-of-the-art models for these tasks continue to improve.

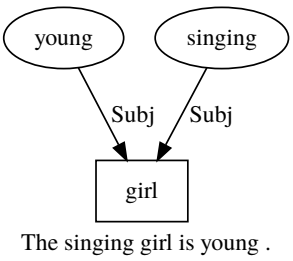
Automation and Domain-Generality Our approach is fully automated and although the generated ASGs can usually only handle a specific set of constructions, our process for generating the ASGs is domain general.

8.1.2.2 Limitations

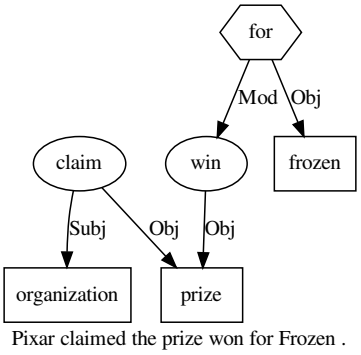
Preposition Senses and Discourse Relations One of the most significant limitations of our approach is that we do not attempt to disambiguate between different senses of prepositions and discourse connectives. As shown in Table 8.1, this kind of disambiguation is attempted in relatively few existing schemes, however we note that it is vital for good semantic



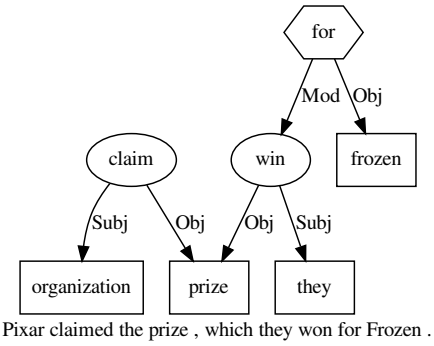
(a) Knowledge graph generated for copular form.



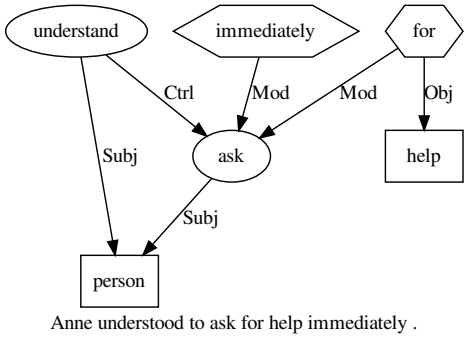
(b) Knowledge graph generated for copular form.



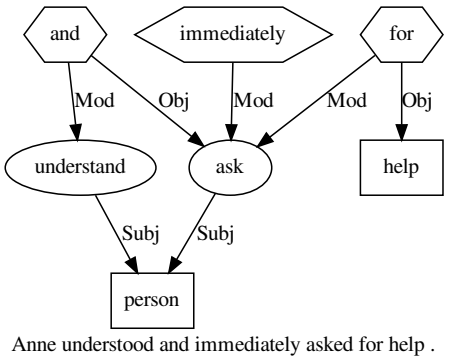
(c) Knowledge graph generated for a passive construction.



(d) Knowledge graph generated for a non-passive construction.



(e) Knowledge graph generated for control.



(f) Knowledge graph generated for coordination.

Figure 8.4: Demonstration of canonical forms. These examples show how we handle structures such as copular form, passives, relative clauses, control and coordination, generating the same structures in our knowledge graphs, even though the dependency structures may differ significantly.

understanding. For example, we would not be able to match the causal sense of *since* in *I haven't left the house* ***since*** *it's dangerous outside* matches with *because* in *I haven't left the house* ***because*** *it's dangerous outside*, or the temporal sense of *since* in *I haven't left the house* ***since*** *a week ago* with *for* in *I haven't left the house* ***for*** *a week*, although we can handle some basic cases (e.g. *in*, *into* and *inside*) through synonymy which is encoded in background knowledge. We chose to use lemmas directly instead of preposition senses as we found that incorrectly assigned prepositions would make learning tasks unsatisfiable, unless we were able to find another example that had been assigned an incorrect preposition in a similar way, and because lemmas in similar context often have the same sense anyway.

Cross-POS Representations Another significant limitation of our representation, but one which is shared by all of the representations considered in Table 8.1 save AMR, is its inability to match the same concept when it is expressed using different derivational forms with different parts of speech. For example, consider the three sentences:

She fears heights.

She has a fear of heights.

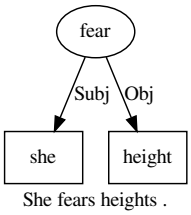
She is fearful of heights.

Clearly all of these sentences have the same meaning and should have one canonical form, but our semantic parsing method is so closely tied to syntax that we do not pickup their similarity. Indeed, even the individual words (the verb *fears*, noun *fear*, and adjective *fearful*) get assigned completely different representations, as can be seen in Figure 8.5. Solving this issue in particular is tricky for our approach, given that we must closely couple semantic composition to syntactic composition, and that to our knowledge, there is not an existing knowledge base which links such derivational forms in a meaningful way.

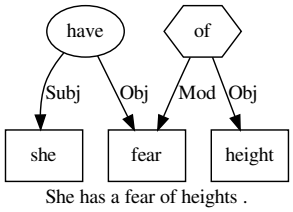
Performance and Domain-Specificity Finally, although the process for generating ASGs is domain specific, the generated ASGs themselves can only parse constructions which have been seen during their generation. This makes them suitable and reusable in some cases where texts are expected to take similar forms (such as in many domain-specific semantic parsing problems, and in the bAbI tasks dataset, for example), or suitable for one-off use when constructed based on the texts they are intended to parse. However, generating domain-general ASGs for English language would be an incredibly difficult task due to the sheer number of constructions it would need to support and the size of the vocabulary. Indeed, even ASGs which capture a relatively small number of sentence forms can perform very poorly, as currently the implementation we use for solving ASGs relies on a greedy approach for parsing which does not scale well when there are many possible production rules to choose from.

8.2 Learning

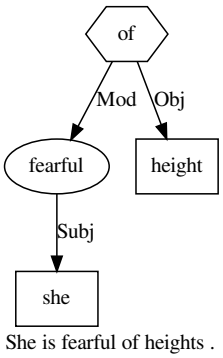
In this section, we assess the capability of ASG induction to learn useful commonsense rules that extend our base ASGs, and allow us to solve coreferencing problems. We investigate our performance on bAbI tasks 11 and 13, as well as on the Winograd schema challenge.



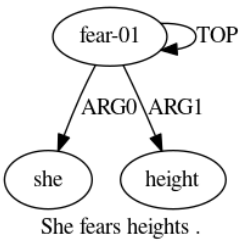
(a) Our representation of the verbal form.



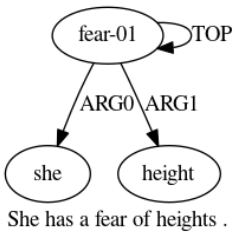
(b) Our representation of the nominal form.



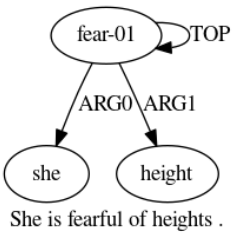
(c) Our representation of the adjectival form.



(d) AMR representation of the verbal form.



(e) AMR representation of the nominal form.



(f) AMR representation of the adjectival form.

Figure 8.5: WinoGrammar and AMR representations for phrases using different forms related to the word "fear." Note how our approach assigns different representations to each form (even for the individual words: the verb *fears*, noun *fear*, and adjective *fearful*), while AMREager produces a canonical form for all three examples.

1 Sandra went back to the bathroom.	
2 Afterwards she travelled to the office.	
3 Where is Sandra?	*office*
4 John went to the garden.	
5 Following that he went to the bedroom.	
6 Where is John?	*bedroom*
7 Mary went to the office.	
8 Then she journeyed to the garden.	
9 Where is Mary?	*garden*
10 John journeyed to the kitchen.	
11 Following that he travelled to the bedroom.	
12 Where is Mary?	*garden*
13 Mary went to the kitchen.	
14 Then she went to the bedroom.	
15 Where is Mary?	*bedroom*

Table 8.3: An example from bAbI task 11.

8.2.1 Performance on bAbI Tasks

Motivation In our first experiment, we attempt to learn the knowledge required for the coreferencing tasks (tasks 11 and 13) from the bAbI dataset. The bAbI tasks are very well-suited to inductive learning approaches due to their fixed vocabulary, lack of noise and generally straight-forward sentence structure. Furthermore, inductive logic programming has already been applied to many of the bAbI tasks (Chabierski et al., 2017), including tasks 11 and 13 (Mitra and Baral, 2016).

Methodology Let us now explain how we setup our learning tasks for the bAbI conferencing tasks. First, recall the example format for bAbI tasks 11 and 13. Each example consists of 10 sentences and 5 questions. An example is shown Table 8.3.

Our learning tasks are generated as follows:

1. **Generate base ASG.** Firstly, we generate a ASG based on a set of training examples from bAbI task 11. Specifically, for each example, the ten sentences are joined together as a single text and passed to our ASG generation pipeline, while the questions are ignored entirely. All sentences from each example are passed in at once, so that our ASG will encode that each sentence follows from the previous (by means of a $next\_event$ modifier), and can capture which events happen at which time-steps (by means of a $happens$ predicate). We make only a single change to our ASG generation pipeline, which is that proper names are assigned their values directly as their identifiers, so that we can capture that, for example, *Mary* always refers to the same entity. For the example in Table 8.3, sentences 1 and 2 would generate a representation including the

literals:

$$event(e1, go, sandra) \wedge nominal(sandra, person) \wedge modifer(e1, to, bathroom)$$
$$\wedge event(e2, travel, she) \wedge nominal(n1, she) \wedge pronoun(n1) \wedge modifier(e2, to, office)$$
$$\wedge modifier(m1, next\_event, e1, e2) \wedge happens(e1, 1) \wedge happens(e2, 2)$$

2. **Translate examples.** Each training example is translated into a logical form. It is important to note that we do not parse the question itself — we instead transform each question into two binary choices. For example, question 3 in Table 8.3 would, in effect, require the learner to come the conclusion $at\_location(sandra, office, 2)$ ***and*** reject the conclusion $at\_location(sandra, bathroom, 2)$. As such, each individual example is very long and has five inclusions and five exclusions which must be explained by the learner.

3. **Add background knowledge.** We add hand-written background knowledge. This consists of the facts $time(0..10).$ and the single rule which encodes the successor relation:
$$inc(T, T + 1) \leftarrow time(T), time(T + 1).$$

4. **Add mode declarations and bias constraints.** We add a set of hand-written mode declarations. The mode declarations allow the learner to learn rules with $at\_location($ $Person, Location, Time)$ or $coref\ (Pronoun, Person)$ at their head. They allow body literals to specify events and modifiers, as well as use the inc predicate from the background knowledge, or to use the learned $at\_location$ and $coref$ predicates.

   We also add a set of hand-written bias constraints to reduce the size of the hypothesis space. In total there are nine of them. These are simply used to restrict some combinations of head and body literals. For example, we do not allow the $at\_location$ or $coref$ predicates to be used in the body of a rule which has a $coref$ predicate as its head.

5. **Train.** The base ASG, examples, background knowledge, mode declarations and bias constraints are combined into a single learning task and the task is processed using the existing ASG solver binary. A maximum of 5 body literals and 6 variables were allowed per learned rule.

6. **Test.** To test our process, we generate a set of new examples, from bAbI's test data, in the same way as our training examples are generated. We then check that the hypothesis generated by the training examples generates the correct inclusions and exclusions for the test examples. We tested on 100 examples for each task. To allow us to reuse the same example generation procedure, each example was considered answered correctly only if all five of its questions were correctly answered.

Since our goal is to test the limits of *learning* on top of ASGs for natural language, our approach differs significantly to previous approaches. Most notably, previous approaches (Mitra and Baral, 2016; Chabierski et al., 2017) tend to provide some event calculus axioms in their background knowledge, taking the form:

$$holdsAt(F, T + 1) \leftarrow initiatedAt(F, T).$$
$$holdsAt(F, T + 1) \leftarrow holdsAt(F, T), \text{not } terminatedAt(F, T).$$

No. of examples	Training time (s)	Accuracy, task 11	Accuracy, task 13
1	24	32%	87%
2	168	100%	100%
5	690	100%	100%

Table 8.4: Summary of results for the bAbI tasks dataset.

The first rule states that a fluent holds after it is initiated, while the second states that a fluent continues to hold until it is terminated. These help the learner by handling the recursion needed to express the fact that a fluent will continue to hold. In order to increase the difficulty of our learning task, we do not include such rules.

Results The results of our experiment are summarised in Table 8.4. The main difficulty for our approach is finding a question that forces the learner to learn that somebody stays in the same place unless they move. In Table 8.3, this would be a question like question 12. When running our experiment, the first example did not have any such questions, and so the learner was able to satisfy the training example with a hypothesis which was insufficient for most testing examples.

After a small number of examples (less than 2 on average), our approach settles on the hypothesis:

$$
\begin{aligned}
coref(P, C) \leftarrow & pronoun(P), modifier(\_, next\_event, E1, E2), \\
& event(E1, travel\_v\_02, C), event(E2, travel\_v\_02, P). \\
at\_location(P, L, T) \leftarrow & event(E, travel\_v\_02, C), modifier(\_, to, E, L), happens(E, T). \\
at\_location(P, L, T') \leftarrow & at\_location(P, L, T), inc(T, T'), \\
& \text{not } event(E, travel\_v\_02, P), happens(E, T). \\
at\_location(P', L, T) \leftarrow & at\_location(P, L, T), coref(P, P').
\end{aligned}
$$

The first rule asserts that a pronoun (specifically, a pronoun which is moving) refers to the person who acted (specifically, the person who moved) in the previous event. In the context of the bAbI coreferencing tasks, this is the correct rule to be learned, and in previous symbolic approaches to the task, this was the full extent of what was learned (Mitra and Baral, 2016). We additionally learn three rules which specify a person's location:

1. If somebody moves to a location, then they are at that location.
2. If somebody is at a location, and they are not known to move, at some time-step, then they are still at that location at the next time-step.
3. If a pronoun is marked as at a location, then the person corresponding to that pronoun's referent is at the location.

This hypothesis correctly answers not only bAbI task 11, but also task 13, compound coreference, since our base ASGs automatically capture coordination, producing events for both nominals.

Discussion It is important to note that the motivation of this work is to produce a proof-of-concept for learning complex rules from natural language examples using ASGs, and as such we have manually defined some parts of the learning task, as discussed in our methodology. Our work is therefore not directly comparable, even to existing symbolic approaches. However we note that both neural approaches and the previous symbolic approach to bAbI tasks 11 and 13 (Mitra and Baral, 2016) also achieve 100% accuracy, although they do not state how many of the 200 example texts they use. Additionally, we note that all of the main limitations of our approach are ones which we share with the existing symbolic approach (Mitra and Baral, 2016): firstly, both approaches specify their hypothesis spaces manually; secondly, we both develop models for individual bAbI tasks, and not a single model capable of handling the full set of 20 tasks; and finally, the learned rules are event dependent.

Importantly, evaluating our performance on bAbI tasks 11 and 13 led us to the following observations, which motivated our approach to the WSC dataset:

1. **It is possible to learn complex hypotheses on top of our generated ASGs for natural language.** The hypothesis learned for the bAbI coreferencing task is relatively complex, learning multiple rules, some of which have recursive definitions (*at_location*), and demonstrates non-observational predicate learning through its definition of *coref*. Furthermore, the accuracy of the approach demonstrates that our ASG can parse examples from bAbI tasks 11 and 13 perfectly, and our ability to answer both tasks 11 and 13 by training only on task 11 demonstrates the capability of our semantic parses to recognise similar structures. Overall, this confirms our ability to learn (complex) commonsense rules from natural language examples using our ASGs, and so helps to validate our hypothesis that ASG induction might be useful for the WSC.
2. **Optimising the size of the hypothesis space is important.** The most significant challenge of bAbI tasks 11 and 13 for our approach, was that requiring the learner to explain a person's location in addition to resolving the coreference resulted in a very large hypothesis space that would take hours to search. The main learning was therefore the necessity of choosing variable and constant types to minimise the number of possible combinations, combined with the use of effective bias constraints. In particular, the use of bias constraints in this task led to a reduction in the size of the search space from some 13 982 rules down to 1063.

8.2.2 Performance on the Winograd Schema Challenge

In this subsection, we evaluate the performance of our approach on the WSC273 task. We firstly discuss the accuracy our approach achieves, before investigating more thoroughly how the selection of examples and our hypothesis space influences performance. We then evaluate the quality of our explanations, and finally outline the main strengths and weaknesses of our approach.

8.2.2.1 Accuracy

We ran all 273 Winograd schemas through our system. In total, **91** schemas (33%) were answered correctly and **5** schemas (2%) were answered incorrectly, while 177 schemas (65%)

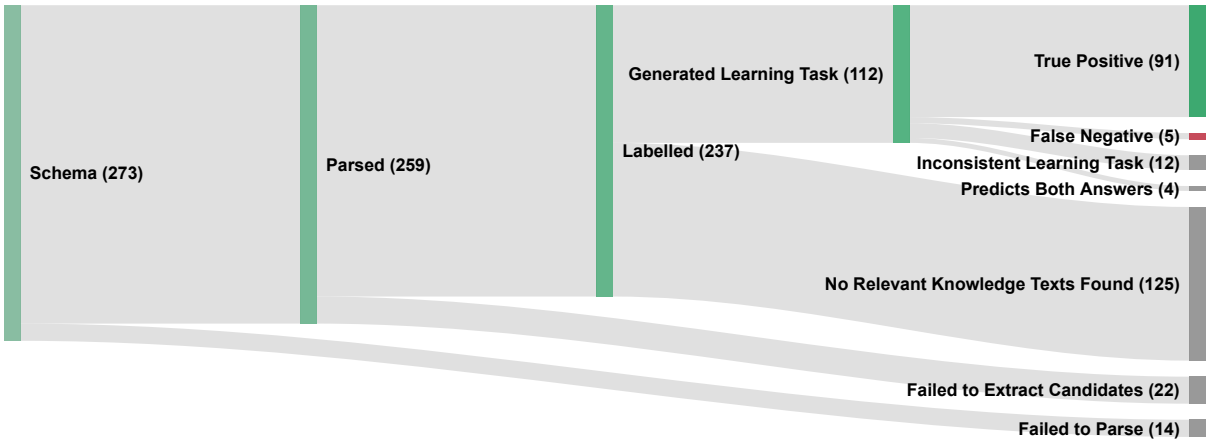


Figure 8.6: Full classification of Winograd schemas according to their result, or the reason why they were left unanswered.

were left unanswered. This gives our approach a precision of **0.95** and a recall of **0.33**.

It is important to note that we will not make a prediction if (i) we cannot find any relevant knowledge, (ii) we find contradictory examples that cannot be explained by a hypothesis in our hypothesis space, or (iii) we find evidence to support both candidates and no evidence to reject either of them. This explains how we are able to achieve such high precision — any example which manages to overcome all of these hurdles will very likely be answered correctly.

While our accuracy is 33.3% — i.e. less than chance for a binary decision problem, it is important to note that this result cannot be compared fairly against existing machine learning-based approaches which attempt the full dataset. Indeed, just by making a random guess for the 177 unanswered schemas, we should expect on average to get a further 88.5 schemas correct. As such, this extended approach could be expected to achieve 65.8% accuracy on the full WSC, and would therefore be roughly comparable in accuracy to many recent language-model based approaches such as an ensemble of 14 pre-trained LMs trained by Trinh and Le (2018) (63.7%), or OpenAI's GPT2 (Radford et al., 2019) (70.7%).

Sources of Errors Because of the amount and complexity of pre-processing required in order to generate a learning task for a Winograd schema, and the noisy nature of examples which are sourced from the internet, there are many reasons why a schema might be left unanswered, or answered incorrectly. Figure 8.6 provides an overview of how common each of these reasons are. Let us now investigate more closely the main sources of errors:

1. **Parsing failure.** Approximately 5% of schemas do not generate a base ASG. This is because we currently have no translation for certain dependencies, and fall back to a default translation. Since this default translation is often incorrect, any schema containing an unsupported dependency is currently ignored. The most significant issue is that we currently do not translate the `expl` (expletive) dependency, which occurs in fourteen schemas, including, for example:

   ***There is** a gap in the wall. You can see the garden through it.*

Defining the correct translation is usually a straightforward process, and it would likely not be too difficult to resolve this issue in future work.

2. **Labelling failure.** For approximately 8% of schemas, we fail to determine the nominals in the schema's knowledge graph which correspond to the candidates. As we note in Subsection 7.2.1, this is because our labelling approach for Winograd schemas is currently very primitive. For example, we do not match the candidates *old house* and *new house* to *his house* and *a new one* respectively for the schema:

   *Joe has sold **his house** and bought **a new one** a few miles away.*
   He will be moving out of it on Thursday.

   since the text does not match. Again, solving this problem is largely an issue of engineering work.

3. **Knowledge hunting failure.** The most significant bottleneck to our performance is lack of knowledge texts — almost half of all schemas (46%) fail at the knowledge hunting stage. However, we note that previous knowledge hunting approaches report similar results — for example, Sharma (2019) finds knowledge for exactly 50% of the 240 problems attempted. An example of a schema that fails to match any knowledge text is:

   ***Fred** is the only man alive who still remembers **my father** as an infant.*
   *When Fred first saw my father, **he** was twelve years old.*

   The knowledge required to handle this example (that twelve years old is too old to be an infant) is both highly specific and not knowledge that we can expect to be written down anywhere. In order to be able to solve schemas like this one, we would need to be able to retrieve and make use of significantly more general and variable texts than we do currently, which is not a straight-forward task.

4. **No hypothesis.** Of schemas which successfully generate a learning task, 11% are not solved because there is no hypothesis in the search space capable of explaining the examples, making this the most common cause of failed learning tasks. As an example, let us consider the schema:

   ***Joe** paid **the detective** after **he** received the final report on the case.*

   The knowledge texts which are selected for this schema include:

   *At dinner **Felicia** decided to pay for **Elena**'s meal*
   *because **Felecia** had received a large bonus today.*

   and

   *At dinner **Felicia** decided to pay for **Elena**'s meal*
   *because **Elena** had received a large bill today.*

   Since the hypothesis space has no way to distinguish between the *bonus* and the *bill*, it is not able to explain the fact that *the person who pays is the person who receives* in

the first example, but not the second. We believe that these cases could often be solved by weighting examples. In this particular example, there are a total of five examples found by the knowledge hunter — four of which are explained the hypothesis that *the person who pays is the person who receives*. Furthermore, the example that generates the inconsistency is determined to be the least relevant of the five examples, so if the weighting were based on relevance, there would not be a significant cost applied for its mislabelling.

5. **Unusable hypothesis.** There are four learning tasks (4%) which generate a hypothesis that cannot be used, as it predicts both answers. Consider, for example the schema:

   Jane *knocked on the door, and* ***Susan*** *answered it.* ***She*** *invited her to come out.* (8.1)

   The knowledge hunter finds the text:

   Jeremiah *knocked on the door and invited them to come out for a picnic.* (8.2)

   and:

   When I knocked, ***Mrs. M*** *answered the door and invited me in.* (8.3)

   These two examples can be explained by the hypothesis:

$$\begin{aligned} may\_refer\_to(P, C) \leftarrow & event(\_, invite, P, \_, \_), event(\_, answer, C, \_). \\ may\_refer\_to(P, C) \leftarrow & event(\_, invite, P, \_, E), modifier(\_, out, E), \\ & event(\_, knock, C, \_). \end{aligned}$$

   Let us note how the first rule states that *the person who answers is the person who invites*, which correctly answers (8.3) and does not apply to (8.2), and returns the (incorrect) answer of *Susan* when applied to (8.1). Meanwhile the second rule states that *the person who knocks is the person who invites somebody to _ out*, which answers (8.2) correctly and does not apply to (8.3), but instead returns the (correct) answer *Jane* when applied to (8.1). There is little we can do to solve these cases except attempt to source more knowledge in the hope of learning something which excludes one of the answers.

6. **Incorrect hypothesis.** A final 4% of learning tasks fall into the worst error case — by predicting the wrong answer. This generally happens when the learner has not seen any example with enough overlapping context with the schema. An example of a schema which is answered incorrectly is:

   The lawyer *asked* ***the witness*** *a question , but* ***he*** *was reluctant to repeat it .*

   This schema cannot be answered correctly as we find no example where the question-asker is reluctant to repeat themselves. For example, the most relevant text found by our knowledge hunter is:

   Someone once asked ***me*** *a strange question . But* ***I*** *am reluctant to repeat it.*

   which leads us to the incorrect hypothesis that *the person who is asked is the person who is reluctant*. In this case, the learner perhaps needs an example with the precise context of a *lawyer* being reluctant to repeat themselves in order to better specify its hypothesis.

Consistency In order to evaluate the consistency of an approach to the WSC, Trichelair et al. (2019) propose testing against a "switched" version of the dataset, in which, where possible, the positions of the two candidates in the schema are swapped. The idea is that, for example, when the schema *Jim yelled at Kevin because he was so upset* is changed to *Kevin yelled at Jim because he was so upset*, we should check whether our answer switches as expected. The proportion of predictions that change is called the *consistency score*.

131 of Winograd schemas can be treated in this way, of which we answer 54. For these 54 schemas, our answer *always* switches. This is because we learn the exact same set of rules for the switched dataset, and there is currently no schema for which we learn a rule which specifies the candidate itself. This gives our approach a consistency score of **100%**. In fact, even if we were to make a random choice on the remaining 77 schemas that were not answered, we would still expect to get a consistency score of 71% on average.

Consistency is a property which we share with existing knowledge hunting and reasoning approaches (Emami et al., 2018; Sharma, 2019), which also score 100% on this metric. On the other hand, machine learning based approaches are notoriously inconsistent when answering the WSC, and often swap their answers even when the candidates correspond to proper nouns. For example, the best performing model reported in (Kocijan et al., 2019b), which is based on fine-tuning BERT, achieves a consistency score of just 55%, while BERT without fine-tuning achieves 44%.

Performance on Problem Classes Given that our approach answers only one third of Winograd schemas, a key question is whether we are simply answering the *easiest* third of the dataset — perhaps it is the case that the best we could possibly expect to do on the remaining two thirds of the dataset is barely better than chance — or whether we are actually answering schemas that other approaches find difficult — in which case ensembling our approaches may actually yield significantly improved performance. In order to answer this question, we investigate our performance on schemas which are (a) ambiguous, i.e. schemas which even humans have difficulty answering, (b) associative, i.e. schemas with strong statistical hints, and (c) answered by current state-of-the art approaches.

- **Ambiguous schemas.** The average human performance on the questions which we answer correctly is 95%, while the average human performance on the questions we answer incorrectly is just 84%, compared to a dataset-wide average of 92% (Bender, 2015). This seems to suggest that our approach mimics human performance to some extent — our approach struggles on the problems on which humans struggle. However it is important to bear in mind that the number of examples, in particular false negatives, is very small, and so this result should be taken with a grain of salt.

- **Associative schemas.** The WSC contains 37 associative schemas. These schemas are much easier for language models than non-associative schemas: e.g. fine-tuned BERT achieves 81% accuracy on the associative set of schemas, compared to 70% on the remainder (Kocijan et al., 2019b). Our approach, on the other hand, actually does particularly poorly on associative schemas — we answer only 7 of them, of which we get 6 right and 1 wrong, therefore achieving a precision of 0.86 and a recall of 0.16, both of

Knowledge Sources	Precision	Recall
WinoGrande (Sakaguchi et al., 2020) only	0.89	0.11
Web Search Examples (Sharma, 2019) only	0.88	0.22
Both sources	0.95	0.33

Table 8.5: A comparison of how our approach performs when restricted to a specific knowledge source.

which are significantly lower than our dataset-wide results. This is because associative schemas rely on knowledge of the identity of the candidates: for example, in the schema *I'm sure my map will show this building, it is very famous*, we need a knowledge text to express that buildings can be famous (while maps cannot). However in practice we rarely find knowledge texts with similar candidates, making such problems very difficult.

- **Schemas answered correctly by WinoGrande.** The current state-of-the-art approach, which fine-tunes RoBERTa using the WinoGrande dataset (Sakaguchi et al., 2020), answers 245 schemas correctly (90%), and 28 incorrectly. Of the 28 schemas which are answered incorrectly by WinoGrande, we answer 7 correctly, and of the 5 schemas we answer incorrectly, WinoGrande gets only 2 correct. This helps to reaffirm our hypothesis that we do not just answer the easiest third of schemas, and that our approach in fact is good at answering a different set of schemas than language model-based approaches. A secondary result of this finding is that, by using our approach for an initial classification, and using the WinoGrande model only if our approach does not provide an answer, 5 additional schemas would be answered correctly, corresponding to an 18% reduction in error compared to WinoGrande alone.

8.2.2.2 Effects of Example Selection and Hypothesis Space Structure

In this project, there are two fundamental processes which affect how our learning process performs: the selection of examples and the generation of the hypothesis space. We now ask how significant each of these processes are are, both in terms of the accuracy we are able to achieve and the computational cost of the learning tasks. In particular, we are interested in how important each of our knowledge sources are, as well as how many examples are necessary to learn commonsense concepts. We also look to investigate whether the ability to learn exceptions is helpful, and whether our optimisations to restrict the hypothesis space are necessary and effective.

Knowledge Sources We discuss our choice of knowledge sources in Subsection 7.2.2. In order to investigate the importance of each of these sources, and the availability of knowledge in general, we reran our experiments after removing all examples from each of them. Our results are shown in Table 8.5. Our two main findings are as follows:

- There are actually only a very small number of schemas (13 in total) which use knowledge from both sources. In particular, note how the recall of our approach using both

Max. Examples	Avg. Parsing Time (s)	Avg. Learning Time (s)	True Positives	False Negatives	Precision	Recall
1	5.0	49.4	92	12	0.885	0.337
2	95.8	55.0	91	5	0.948	0.333
3	168.8	56.1	88	3	0.967	0.322

Table 8.6: A comparison of how our approach performs according to the limit on the number of examples.

sources is almost the sum of the recall of our approach using each of the two knowledge sources. This demonstrates the importance of having a large number of examples from a variety of sources. Some "toy" concepts, such as that s*mall things might not fit* appear regularly in the WinoGrande dataset, while other concepts, such as that *the thing/person that less popular cedes to the thing/person that is more popular* are quite specific and are more likely to be found in, say, news articles by means of a web search.

- When knowledge exists from both sources, we are significantly more likely to either answer the example correctly, or produce an inconsistent learning task. This explains why the precision of the approach improves when both knowledge sources are used. This mostly demonstrates the need for multiple examples per schema in order to force the learner to develop a precise hypothesis that takes into account the full context of the example.

Number of Examples Used We note in Subsection 6.1.2, that after scoring the relevance of each knowledge, we choose the top-N knowledge texts as examples for the learning task. By default, we have let $N = 2$ up until this point. We now investigate the effect of changing the limit on the number of examples. Table 8.6 shows the results of this experiment. We point out the following observations in particular:

- As you would expect, increasing the number of examples leads to a decrease in recall. This is because additional examples might create contradictions that cause a learning task to become unsatisfiable. On the other hand, adding extra examples often requires the learner to make its hypothesis more specific, and take additional context into account. These more specific hypotheses are more likely to answer a schema correctly, and so the precision increases. This effect is most notable when increasing from 1 example to 2, since a single example cannot, on its own, force the learner to take any context into account.

- Overall, the effects of adding additional examples are not very significant in terms of accuracy. We suggest that this is because each single example is annotated with quite a lot of semantic information, and so often might be sufficient on its own to learn the correct general concept. This property closely mirrors a property of human *learning by language*, as noted by Levesque (2017), in that "the data might need to be seen or heard just once to do its job." On the other hand, we note that anecdotally, the quality

of learned rules tends to increase as more examples are added, since the learner is forced to take into account additional context.

- Finally, let us note the performance implications of additional examples. In practice, it is not possible to process the full WSC dataset with 5 or more examples, since the ASG learning tasks quickly become computationally infeasible. But upon deeper investigation, we believe this is largely due to the inefficient greedy parsing method which is used by the ASG solver binary we depend upon. Parsing additional examples, which may take very different syntactic forms, requires a much larger and more complex grammar which causes the parser to explore a huge search space. Meanwhile, we note that the learning time does not increase significantly. This demonstrates that our efforts to reduce the size of the hypothesis space are working, and that we could likely handle many more examples if the parsing was optimised.

Learning of Exceptions In Subsection 6.2.2, we motivated our approach to learning exceptions by sketching an example of a learning task that could *only* be explained by allowing exceptions. To investigate how common such scenarios are, and by extension, the utility of learning exceptions, we ran an experiment in which learning of exceptions was not allowed. We found that the number of true positives fell by 16, to 75 schemas, while the number of false negatives remained the same. This result confirms our hypothesis that the learning of exceptions is necessary for good performance: in our approach, it increases the proportion of learning tasks which are satisfiable by over 14%.

Hypothesis Space Optimisations In Subsection 6.2.2, we describe four optimisations which we make in order to reduce the size of the search space. In order to evaluate their effectiveness, we ran the learning pipeline with each of them disabled, and measured the size of the hypothesis space produced for each learning task. Figure 8.7 shows the results of this experiment. Our findings are as follows:

- We find that by far the most significant optimisation is the pruning of the search space based on the examples. Without this optimisation, we were not able to get the learning tasks to run.
- Imposing a depth limit on the hypothesis space affects only a small number of problems with particularly large hypothesis spaces, but where there is an effect, it is significant. In particular, four schemas which initially produced hypothesis spaces too large to generate became feasible after applying a depth limit of 3.
- Removing rules with negation as failure has a small impact on the size of hypothesis space, reducing the number of rules on average by 16% from 702 to 591. It also made one additional task feasible. We found that, presumably due to our ability to learn exceptions using the $cannot\_refer\_to$ predicate, no previously-solvable schema was made unsolvable by this optimisation.

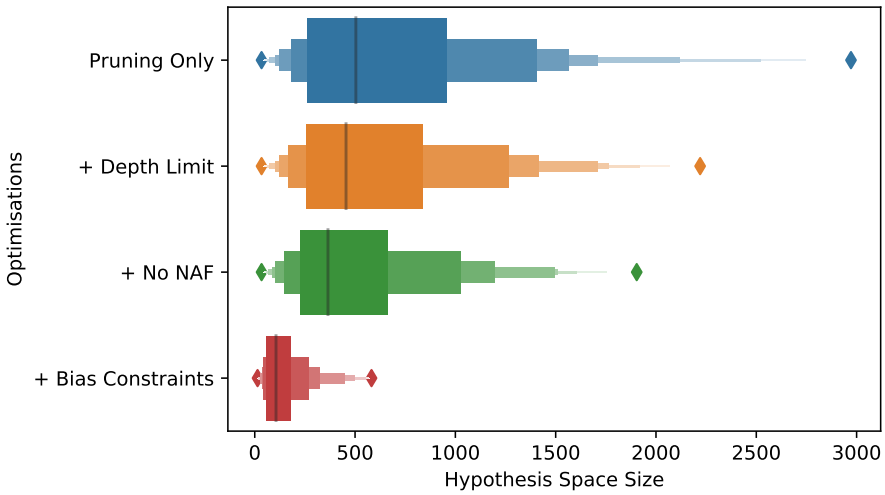


Figure 8.7: The effect of our optimisations on reducing the size of the hypothesis space. We note that, when using only the pruning optimisation, four tasks generated such large hypothesis spaces that the process generating the search space was killed. This plot uses results only from the tasks which generated hypothesis spaces under all settings.

- Finally, our use of bias constraints has a significant impact, reducing the size of the hypothesis space almost 4-fold from 590 rules to 148 on average. The use of bias constraints ruled out four previous hypotheses that led to correct predictions and three that led to incorrect predictions.

In addition to their effect of reducing the hypothesis space and making our learning tasks feasible to solve in a reasonable amount of time, we note that removing negation as failure and adding bias constraints to remove unlinked arguments also resulted in higher quality explanations that are easier to understand.

8.2.2.3 Quality of Learned Rules

The most significant difference between our approach and previous approaches is our ability to generate an *explanation*, based on the hypothesis that is learned. Since, as far as we are aware, there is no published existing work on generating explanations for Winograd schemas, and very little work even on determining what the valid explanations are, we investigate the quality of our explanations individually by hand. For each schema, we generate a table containing the premise, question, our answer and automatically generated explanation, and classify the explanation into one of three categories: *ideal*, *acceptable*, or *incorrect*, in addition to adding a short comment on the classification. We provide an example in Table 8.7.[1]

The example in Table 8.7 has been marked as *acceptable*. This classification is for explanations which make sense overall, but may be missing some context which is relevant to the answer. For this example, an *ideal* explanation might be "something may not fit *because* it is large"

[1]The full set of answers, explanations and comments has been made available online at https://www.doc.ic.ac.uk/~js4416/winograd.html.

Premise	The trophy doesn't fit into the brown suitcase because it is too large.
Question	What is too large?
Answer	Trophy (Correct)
Explanation	The thing which doesn't fit is the thing which is large.
Comments	*Acceptable.* This hypothesis could be improved by taking into account causality.

Table 8.7: An example of an explained answer to a Winograd schema, and its classification and comment.

— i.e. it should take into account the causality of the concept. Meanwhile, an explanation is classified as *incorrect* if it does not make sense — it may be missing context that is essential, or uses context that isn't relevant. An example of an *incorrect* explanation might be "the thing which is large is not the thing which is brown." While this explanation would give the correct answer for this schema, it is clearly not a valid explanation.

Using these criteria, we find that **27** schemas produce *ideal* explanations, **55** schemas produce *acceptable* explanations, and the remaining **14** schemas produce *incorrect* explanations. This means that for 86% of schemas which we answer correctly, the explanation is *acceptable* or better. The main observation made during the evaluation of explanations is that our learner often produces quite general explanations — it misses small pieces of context that are relevant, like in the example in Table 8.7. We believe that the solution to this problem involves finding more examples, in particular with more variability, which will force the learner to hone in on more specific explanations.

8.2.2.4 Comparison with Previous Work

Let us now compare our work with existing approaches to the Winograd Schema Challenge. We provide a brief overview of how our approach compares to both previous symbolic and neural approaches in Table 8.8. In particular, we note how our accuracy is roughly comparable to existing approaches, with the exception of RoBERTa fine-tuned on WinoGrande (Sakaguchi et al., 2020), but we are the first approach to generate explanations. In this subsection, we will look at some of the technical differences between our approach and existing knowledge hunting-based approaches.

Knowledge Hunting With regards to the knowledge hunting method itself, both Sharma et al. (2015) and Emami et al. (2018) use syntax-based approaches, generating queries which are used to filter out irrelevant texts.

The approach proposed by Sharma et al. (2015) builds queries simply by swapping out nominals in the texts with wildcards. For example, the text "*the trophy* would not fit into *the brown suitcase* because *it* was too large" would generate the query ".* would not fit into .* because .* was too large". While this approach is very straight-forward to implement and results in very

Approach	Work	Coverage	True Positives (Precision)	Training Examples	Automated?	Able to use background knowledge?	Produces Explanation?
Logical Reasoning	Schüller (2014)	8 (3%)	8 (**100%**)	**0**	No	**Yes**	No*
	Sharma (2019)	200 (73%)	200 (**100%**)	**0**	**Yes**	**Yes**	No*
Knowledge Hunting	Sharma et al. (2015)	52 (19%)	49 (94%)	**Up to 1 per schema**	**Yes**	No	No
	Emami et al. (2018)	189 (69%)	106 (56%)	Many per schema	**Yes**	No	No
Language Models	Trinh and Le (2018)	**273** (**100%**)	147 (54%)	~ 100 000	**Yes**	No	No
	Kocijan et al. (2019b)	**273** (**100%**)	198 (73%)	~ 2.4 million	**Yes**	No	No
	Sakaguchi et al. (2020)	**273** (**100%**)	**245** (90%)	~ 41 000	**Yes**	No	No
Inductive Learning	Ours	96 (35%)	91 (95%)	**Up to 3 per schema**	**Yes**	**Yes**	**Yes**

Table 8.8: A comparison of our approach against existing symbolic (reasoning/knowledge hunting) approaches and the state-of-the-art (language model-based) approaches. Note that the reasoning-based approaches (starred) take the required explanation as an input.

precise matches, it doesn't work so well for long sentences with a lot of context, and often results in few or no results, when there are relevant results available.

The approach proposed by Emami et al. (2018) splits up schemas based on part-of-speech tags. It first tries to find the candidates and pronoun, and then splits the schema into a *context* part (containing the two candidates) and a *query* part (containing the pronoun). Verbs and adjectives within these parts of the schema are then used to generate queries. For example "*the trophy* would not fit into *the brown suitcase* because *it* was too large" would be split into the context "*the trophy* would not fit into *the brown suitcase*" and the query "*it* was too large". It then generates a set of *context* queries such as "would not fit", "fit", and "brown", and a set of *query* queries such as "was too large" and "large". This approach is much more complex, and requires schemas to take an exact format, but produces more flexible queries that are more likely to find useful knowledge texts.

Our approach is instead based on semantics, our queries are generated using a schema's knowledge graph. Unlike the approach proposed by Sharma et al. (2015), this allows us to match on texts which may have a completely different structure than the schema, such as, for example, "The first pick she selected was too large and wouldn't fit in the small keyhole," which their approach would never consider. We also use lemmas and WordNet senses to generate our queries, which allows matching on different conjugations of verbs for example, unlike either of the previous approaches. Furthermore, unlike Emami et al. (2018), since we work from our knowledge graph instead of attempting to split up the text directly, we do not prescribe an exact format on the schema.

The second significant difference in our approach is our ranking of candidates. Sharma et al. (2015) select their (single) final example manually. Meanwhile, Emami et al. (2018) do not prescribe a limit on the number of examples they use, but they do weight their examples based on the length of the query used to match them, and whether or not the context match and the query match occur in the correct order. On the other hand, our ranking approach is automated and is used to select a limited number of examples, based on the amount of shared context.

Learning The process of labelling our schemas with a correct and incorrect interpretation is similar to what has been done in previous approaches. Where our approach differs most significantly from previous knowledge hunting approach is what we do with our labelled examples.

The approach proposed by Emami et al. (2018) matches the labelled candidates back to the schema, and then finds for each example which candidate from the original schema it supports. They then choose the candidate whose supporting examples' weights sum to the greater value.

The approach proposed by Sharma et al. (2015); Sharma (2019) requires an example text which is fully relevant. They have a reasoning process which finds a graph-subgraph isomorphism between the example and the schema which must fully cover the example. Once this has been done, they can choose an answer since the correct candidate in the example should be assigned to a candidate in the schema's knowledge graph.

Our approach replaces this step with inductive learning, in which the learner forms a hypothesis, essentially by specifying which parts of the schema's knowledge graph are relevant to

resolving the coreferencing ambiguity. Importantly, note how, unlike in the approach proposed by Emami et al. (2018), it is possible for us to choose some candidate A over another candidate B, even if there is only one example which supports choosing A and many which support choosing B — i.e. if the example supporting A can be distinguished by some context that is present in the schema and not in any of the examples supporting B, then this will be taken into account by our learner's hypothesis. Furthermore, unlike in the approach proposed by Sharma et al. (2015), we do not require the schema to completely cover the example, but instead apply only very limited constraints on our choice of examples. This allows us to find varied examples forcing us to specify exactly how different subgraphs in the schema's knowledge graph affect the resolution of the pronoun. We note how it is this process which, unlike any previous knowledge hunting approach, allows us to, in the end, return an explanation with our answer.

Producing Explanations Finally, let us briefly discuss related work on the problem of finding explanations for Winograd schemas. To our knowledge, there is only one piece of work which investigates explanations for the WSC (Zhang et al., 2020). Their approach crowdsources explanations to the WSC and they produce a dataset, *WinoWhy*, which "requires models to distinguish plausible reasons from very similar but wrong reasons for WSC questions." They note that *WinoWhy* is a "challenging task", and that their best supervised model, which they train by fine-tuning BERT, achieves just 77.8% accuracy on a two-class classification task. Our approach differs significantly from theirs in that we generate an explanation from a Winograd schema alone, rather than taking an existing explanation and a Winograd schema, and determining if that explanation is valid.

8.2.2.5 Strengths

Finally, let us summarise the strengths and limitations of our learning approach to the WSC, and also point out some of the remaining open questions.

Precision Our approach very rarely produces false negatives. Our precision of 0.95 exceeds that of any previous approach except for reasoning approaches which provide the required explanation as an input.

Explainability Our approach provides explanations for all of its answers, unlike any previous approach. These explanations can be expressed in natural language, and are valid in 85% of cases.

Consistency Our approach answers questions consistently. When the positions of candidates in a schema are swapped, our approach always swaps its answer, thus achieving a consistency score of 100%. In other words, unlike for machine learning approaches, our answer is not affected by people's names or irrelevant properties of objects.

Generality Our approach is not limited to a specific set of schemas, unlike, for example, the approach outlined by Sharma et al. (2015), which only handles two specific forms of example — which they call *event-event causality* and *casual attributive* examples. We are able to produce both very simple hypotheses connecting two events, as well as more complex hypotheses which can express causality, contrast, and ordering of events, by referring to a relevant modifier. For example, the knowledge that being large *may cause* something to not fit can be roughly expressed by learning the rule:

$$may\_refer\_to(P, C) \leftarrow modifier(L, large, P), modifier(\_, because, F, L), \\ neg\_event(F, fit, C).$$

Number and Quality of Examples Our approach can learn the correct commonsense knowledge from very few examples. In fact, for most schemas for which we find knowledge, only a single example is necessary to learn a hypothesis which gets the correct answer. Furthermore, unlike Sharma (2019), we do not manually filter examples, and we can handle examples from a diverse range of sources, including from existing datasets (with labels) or from the web (without labels).

Ability to Incorporate Background Knowledge Unlike machine learning approaches, our approach can easily incorporate prior knowledge by, simply by extending the background knowledge. We demonstrate this by automatically enhancing our background knowledge with properties from ConceptNet, as described in Subsection 5.3.2, however this could also be done manually. This allows us to effectively give the learner hints in a way that is not possible for traditional machine learning approaches. For example, we might tell it that *a butterfly wing is fragile*:

$$modifier(bk, fragile, W) \leftarrow nominal(W, wing), modifier(\_, butterfly, W).$$

in order to help answer the schema *I put the butterfly wing on the table and it broke.*

8.2.2.6 Limitations

Recall Clearly the most significant limitation of our approach is its poor recall, since we leave two thirds of schemas unanswered, although our recall may be artificially increased from 0.33 to 0.66 by making a random choice for unanswered schemas. Alternatively, since we have shown that the schemas handled by our approach are complementary to the schemas handled well by traditional machine learning approaches, our approach may be ensembled with a traditional approach to achieve a recall that is superior to either method alone.

Knowledge Hunting The most significant difficulty for our approach is the process of knowledge hunting. Not only does it limit our recall, since for 46% of schemas, we find no examples, but it also limits the quality of our explanations since we often find only a small number of examples, which can be explained by a very general hypothesis.

Performance In order to be able to parse examples with very different linguistic constructions, we require large and complex grammars. Currently our approach appears to be limited by the performance of the ASG parser for large grammars, which restricts us to a very small number of examples and makes it difficult to generate explanations that make use of a lot of context.

8.2.2.7 Open Questions

Transferability to Other Tasks Our approach should be able to solve any Winograd-like coreference resolution problem, but this is clearly quite a niche use-case. We have also demonstrated the utility of ASG induction in a question-answering setting with our work on bAbI tasks 11 and 13, although those experiments did not use knowledge hunting or our automated approach for generating hypothesis spaces.

However, we have not explored in depth whether our ability to learn commonsense for the WSC is transferable to other commonsense reasoning tasks. This leads to two main questions: firstly, does our evaluation on the WSC give a realistic estimate of our ability to learn commonsense, and secondly, is our approach applicable other tasks?

With regards to the first question, it has frequently been noted that there are many threats to the validity of experiments based on the WSC. As noted by Trichelair et al. (2019) (and others), "the main drawback of the Winograd Schema Challenge is its limited size and the absence of training and validation sets for hyperparameter tuning. As a result, achieving above random accuracy on the WSC does not necessarily correspond to capturing common sense; it could be the result of a lucky draw." However, the benefit of our approach is that we can tell immediately whether we have captured commonsense, not by our accuracy, but by the hypotheses which we produce. Furthermore, our approach has very few parameters compared to machine learning approaches, and we have taken particular care to avoid tuning their values according to performance on the test set.

As to whether we could answer questions from other tasks, it is quite likely that at least some of the work we have done is transferable, as direct knowledge hunting approaches have proven useful in developing approaches to a variety commonsense reasoning datasets (Emami et al., 2018). However, we note that, at the very least, it would be necessary to modify our labelling procedure for examples, in order to be able to cope with other forms of questions.

Chapter 9

Conclusions

In this project, we have investigated the applicability of answer set grammars for natural language understanding, and the applicability of inductive learning (based on those ASGs) for learning commonsense knowledge. Our findings can be summarised as follows:

1. **Answer set grammars are capable of capturing syntactic and semantic patterns in natural language.** Through the ability to parse WSC schemas and examples sourced from the internet, we demonstrate that it is possible for ASGs to be used for simultaneous syntactic and semantic parsing of natural language texts, when these texts are limited to a small enough domain. We find that annotated atoms often allow much more elegant expression of semantic composition when compared to lambda calculus, allowing us to handle constructs such as coordination and passives without any post-processing. When evaluating our approach on semantic parsing for the WSC, we demonstrate that our implementation achieves an accuracy that is comparable with a dependency parse alone but is far more semantically rich. We determine that the most significant challenge for ASG-based approaches for natural language understanding is the difficulty of handling derivationally-related word forms in a consistent manner, however we note that this is a challenge for most grammar-based approaches.

2. **Answer set grammars can encode, and can be used to learn, commonsense knowledge.** In particular, in our bAbI examples, we demonstrate how (a) we can learn complex hypotheses to explain semantics of natural language examples, and (b) an ASG can encode these solutions entirely, without the need for any additional models, acting as both a semantic parser and applying the leaned knowledge at once.

3. **A semantics-based approach to knowledge hunting allows for a diverse set of relevant knowledge texts to be retrieved.** We demonstrate that our approach to knowledge hunting, which generates queries from knowledge graphs, and can match on lemmas, WordNet sense and part-of-speech tags, is able to return texts that shallow syntax-based approaches would never consider. We also demonstrate how the approach can be extended to automatically determine the most relevant texts, in contrast to previous approaches where this was an additional manual step.

4. **Knowledge hunting can be combined with inductive logic programming to extract commonsense rules.** Unlike any previously published approach (knowledge hunting-based or otherwise), this allows us to support our answers to the WSC with precise explanations. We also show how the approach is able to answer schemas that state-of-the-art neural approaches cannot. Our work confirms the conclusions of previous approaches: namely, that knowledge hunting is a difficult task and is the most significant obstacle to good performance — leading to many schemas being left unanswered and others with explanations that are too general due to too few examples.

9.1 Future Work

We finally discuss some of the most interesting and pressing areas for future work. For our work on semantic parsing using ASGs, we propose some simple extensions which would allow us to attempt question answering tasks, and thus quantitatively evaluate our approach against existing broad-coverage semantic parsers, as well as a more significant extension to handle derivationally-related word forms. Meanwhile, for our learning approach for the WSC, we suggest modifications that could be made to the knowledge hunting process to achieve the goal of generating better explanations, rather than just correct answers.

Extending the Lexicon There are several paths for improving the quality of our base ASGs. In particular, we note several deficiencies of our approach in Subsection 5.3.1, which are in relation to the handling of raising, control, tense and quantification. There are also a handful of rarely used dependencies which we do not currently provide a translation for, but could with a small amount of engineering work.

A more involved extension might look at extending the lexicon with additional features such as person, number, tense, mood and aspect for verbs and person, number, gender, case and animacy for nouns. This would allow us to implement constraints to enforce linguistic phenomena such as agreement, and also allow for learning hypotheses which might rely on these features.

With more time, it would also be useful to work on question translation. This is mostly already supported, and just requires one final step to identify the question word. This would allow us to evaluate our implementation against a question-answering dataset such as FreebaseQA, and compare our work quantitatively against other approaches to broad-coverage semantic parsing, outside of the domain of WSC problems.

Generating Canonical Forms for Derivationally-Related Word Forms We found knowledge hunting to be the major bottleneck in this approach: it is very difficult to find knowledge texts that are semantically close enough to the problem in order to allow applicable rules to be learned. Thus a clear direction for future work on this project would to be to expand the possible knowledge sources in order to pick up more examples in the hope that we might, as a result, find more relevant knowledge. However, a more interesting path might be to *make better use of the knowledge that we already have.* This involves extending the semantic parsing capabilities of our ASGs to produce canonical forms in more cases.

Let us again consider the three sentences

She fears heights.

She has a fear of heights.

She is fearful of heights.

which epitomise one of the most significant limitations of our approach, since they all generate different representations and therefore cannot be used effectively as knowledge for one another. We have identified three possible approaches to solve this kind of issue and allow our system to "capture" more knowledge:

1. **Greater use of lexical knowledge.** Using cross-POS lexical knowledge would allow us to handle these cases, however to our knowledge there is currently no source which both provides detailed enough relations with to determine *how* the different words are related.

2. **Use of commonsense knowledge bases.** Many previous approaches have made automated use of commonsense knowledge bases such as ConceptNet, WebChild and ATOMIC, which may be used to generate background knowledge that can match sentences of the form above. However they can contain spurious knowledge and are unlikely to help with highly specific examples.

3. **Use of paraphrasing/textual entailment models.** A final approach might involve using a textual entailment model to see if one construction entails the other. For example, a RoBERTa-based textual entailment model trained on SNLI and MNLI (Liu et al., 2019) is over 99% confident that each of the sentences above entails each of the others. With this information, you could attempt to generate relevant background knowledge or to modify the original examples to get a closer syntactic match. It's not clear, however, how you might detect candidates for such a process in the first place, and it may be tricky to determine what would make a good confidence threshold.

While we have seen that our ASG-based formalism is consistent, robust and accurate, this example clearly demonstrates arguably its most significant disadvantage: it's heavily restricted by the fact that semantics must correspond so closely with syntax. Machine learning approaches can ignore such constraints entirely, often treating semantic parsing simply as a sequence-to-sequence prediction task, and using semantically-rich embeddings from pre-trained language models as input features. As the accuracy of these neural semantic parsers continues to improve (Zhang et al., 2019), it may be worth considering modifying our learning approach to make use of a state-of-the-art AMR parser. The problem and examples would first be processed by the AMR parser, and then the AMRs would need to be translated into a suitable logical form for ILASP (which is not necessarily a straightforward task). However, the most significant challenge of such an approach would probably be to develop a neural parser that is both accurate and consistent enough that similar examples are always assigned similar representations. As we have seen, this property is essential for learning, but is considerably less likely to hold for complex texts when there is no requirement for the parser to observe syntactic structure. In particular, current state-of-the-art AMR parsers have particularly poor performance for semantic role labelling (Zhang et al., 2019), which is an issue as our learning approach relies heavily on the consistency of argument structure across similar examples.

Counterexample Guided Knowledge Hunting One current limitation of our knowledge hunting approach is that it draws inspiration from previous work (Sharma et al., 2015; Emami et al., 2018), whose goal was only to select the correct answer. The goal of our approach is not just to select the correct answer, but also to generate a good explanation. This puts the goal of our knowledge hunting approach at odds with the goals of previous knowledge hunting approaches: in particular, we want to find *diverse* knowledge texts, which force us to contextualise our explanation.

To demonstrate this, let's consider an example.

> Susan knows all about Ann's personal problems because **she** is *indiscreet.*
>
> ---
>
> *Answer 0*: Susan
> *Answer 1*: Ann

The kind of knowledge text a knowledge hunter might find would be:

> A lot of young conservative men knew about **Goodman**'s proclivities — **he** was rather indiscreet.

This example alone teaches the learner that "the person who possesses something is the person who is indiscreet". This hypothesis is clearly insufficient. To improve the explanation, we need an example of the form:

> We knew **she** would tell us all about his personal life because **she** was so indiscreet. (1)

This example contradicts our existing hypothesis and thus forces us to take into account the context that "the person whose possession *is known about* is the person who is indiscreet".

This observation happens to be particularly relevant to our approach, as due to scalability issues, we can use only a small number of examples (say, N). Our current approach chooses the top-N most relevant examples, which often returns the correct answer but can result in learning rules that are too general. This is because for some problems there may be many examples which are all nearly identical to the schema, and thus the knowledge hunter, favouring the most relevant examples, never selects any contradictory examples like (1).

In order to solve this problem, we propose taking inspiration from counterexample guided inductive synthesis (Solar-Lezama et al., 2006) and developing a counterexample guided process for knowledge hunting. In this formulation, the knowledge hunter and the learner are much more closely coupled. The knowledge hunter starts by providing the learner with the single, most relevant example, and the learner generates a hypothesis. This hypothesis is then returned to the knowledge hunter, which applies the hypothesis to all of its candidate examples and discards any examples which are correctly answered. From the remaining candidates, if any, the knowledge hunter selects the most relevant example and the learner is then required to come up with a hypothesis that additionally explains the new example. This process continues until a termination condition is reached. This could be because, say, (a) the

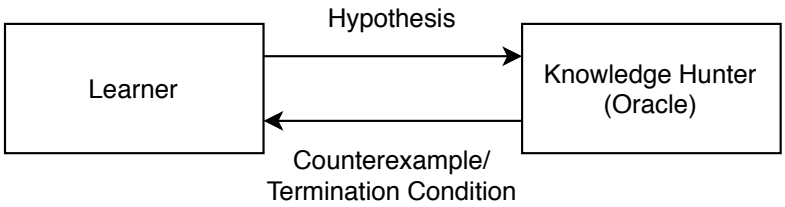


Figure 9.1: A counterexample-guided approach to knowledge hunting. The knowledge hunter would provide examples one-at-a-time to the learner and would ensure that each example is not covered by the existing hypothesis.

knowledge hunter has exhausted its set of examples, (b) the relevance of all of the remaining examples is below a given threshold, (c) the number of examples exceeds a certain threshold (or equivalently the learner times out), or (d) the learner cannot explain the set of examples. Such an approach would make the most of the limited number of examples we may use — *every* example would force the learner to make its hypothesis more specific, thus generating explanations which are more likely correctly incorporate all of the relevant context.

Bibliography

Omri Abend and Ari Rappoport. The state of the art in semantic representation. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 77–89. Association for Computational Linguistics, 2017. ISBN 978-1-945626-75-3. doi: 10.18653/v1/P17-1008. URL https://doi.org/10.18653/v1/P17-1008.

James F. Allen and Choh Man Teng. Broad coverage, domain-generic deep semantic parsing. In *2017 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 27-29, 2017*. AAAI Press, 2017. URL http://aaai.org/ocs/index.php/SSS/SSS17/paper/view/15377.

Daniel Bailey, Amelia J. Harrison, Yuliya Lierler, Vladimir Lifschitz, and Julian Michael. The winograd schema challenge and reasoning about correlation. In *2015 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 22-25, 2015*. AAAI Press, 2015. URL http://www.aaai.org/ocs/index.php/SSS/SSS15/paper/view/10295.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In Stefanie Dipper, Maria Liakata, and Antonio Pareja-Lora, editors, *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, LAW-ID@ACL 2013, August 8-9, 2013, Sofia, Bulgaria*, pages 178–186. The Association for Computer Linguistics, 2013. URL https://www.aclweb.org/anthology/W13-2322/.

Andras Barany. Argument structure and theta roles, 2017.

David Bender. Establishing a human baseline for the winograd schema challenge. In Glass and Kim (2015), pages 39–45. URL http://ceur-ws.org/Vol-1353/paper_30.pdf.

Yoshua Bengio and Yann LeCun, editors. *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL https://iclr.cc/archive/www/doku.php%3Fid=iclr2016:accepted-main.html.

Rajesh Bhatt. Infinitival complementation 1: Control, 2006. URL https://people.umass.edu/bhatt/601-f06/601-f06-l9.pdf.

Blai Bonet and Sven Koenig, editors. *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 2015. AAAI Press. ISBN 978-1-57735-698-1. URL http://www.aaai.org/Library/AAAI/aaai15contents.php.

Johan Bos. Wide-coverage semantic analysis with boxer. In Johan Bos and Rodolfo Delmonte, editors, *Semantics in Text Processing. STEP 2008 Conference Proceedings, Venice, Italy, September 22-24, 2008*. Association for Computational Linguistics, 2008. URL https://www.aclweb.org/anthology/W08-2222/.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018. URL http://arxiv.org/abs/1803.11175.

Piotr Chabierski, Alessandra Russo, Mark Law, and Krysia Broda. Machine comprehension of text using combinatory categorial grammar and answer set programs. In Andrew S. Gordon, Rob Miller, and György Turán, editors, *Proceedings of the Thirteenth International Symposium on Commonsense Reasoning, COMMONSENSE 2017, London, UK, November 6-8, 2017*, volume 2052 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017. URL http://ceur-ws.org/Vol-2052/paper5.pdf.

Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative schemas and their participants. In Keh-Yih Su, Jian Su, and Janyce Wiebe, editors, *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*, pages 602–610. The Association for Computer Linguistics, 2009. ISBN 978-1-932432-45-9. URL https://www.aclweb.org/anthology/P09-1068/.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc V. Le. Semi-supervised sequence modeling with cross-view training. In Riloff et al. (2018), pages 1914–1925. ISBN 978-1-948087-84-1. URL https://www.aclweb.org/anthology/D18-1217/.

Jeff Da and Jungo Kasai. Cracking the contextual commonsense code: Understanding commonsense reasoning aptitude of deep contextual representations. *CoRR*, abs/1910.01157, 2019. URL http://arxiv.org/abs/1910.01157.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. An incremental parser for abstract meaning representation. In *Proceedings of EACL*, 2017.

Donald Davidson. The logical form of action sentences. In Nicholas Rescher, editor, *The Logic of Decision and Action*, pages 81–120. Univ. of Pittsburgh Press, 1967.

Ernest Davis and Gary Marcus. Commonsense reasoning and commonsense knowledge in artificial intelligence. *Commun. ACM*, 58(9):92–103, 2015. doi: 10.1145/2701413. URL https://doi.org/10.1145/2701413.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186. Association for Computational Linguistics, 2019. ISBN 978-1-950737-13-0. URL https://www.aclweb.org/anthology/N19-1423/.

Ali Emami, Noelia De La Cruz, Adam Trischler, Kaheer Suleman, and Jackie Chi Kit Cheung. A knowledge hunting framework for common sense reasoning. In Riloff et al. (2018), pages 1949–1958. ISBN 978-1-948087-84-1. URL https://www.aclweb.org/anthology/D18-1220/.

Ali Emami, Paul Trichelair, Adam Trischler, Kaheer Suleman, Hannes Schulz, and Jackie Chi Kit Cheung. The knowref coreference corpus: Removing gender and number cues for difficult pronominal anaphora resolution. In Korhonen et al. (2019), pages 3952–3961. ISBN 978-1-950737-48-2. URL https://www.aclweb.org/anthology/P19-1386/.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012. doi: 10.2200/S00457ED1V01Y201211AIM019. URL https://doi.org/10.2200/S00457ED1V01Y201211AIM019.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988. ISBN 0-262-61056-6.

Michael Glass and Jung Hee Kim, editors. *Proceedings of the 26th Modern AI and Cognitive Science Conference 2015, Greensboro, NC, USA, April 25-26, 2015*, volume 1353 of *CEUR Workshop Proceedings*, 2015. CEUR-WS.org. URL http://ceur-ws.org/Vol-1353.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007. ISBN 978-0-321-47617-3.

Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, 2019. Association for Computational Linguistics. ISBN 978-1-950737-90-1. URL https://aclweb.org/anthology/volumes/D19-1/.

Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009. ISBN 9780135041963. URL http://www.worldcat.org/oclc/315913020.

Karin Kipper-Schuler. *VerbNet: a broad-coverage, comprehensive verb lexicon*. PhD thesis, Computer and Information Science Department, Universiy of Pennsylvania, Philadelphia, PA, 2005. URL http://repository.upenn.edu/dissertations/AAI3179808/.

Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2676–2686. Association for Computational Linguistics, 2018. ISBN 978-1-948087-32-2. doi: 10.18653/v1/P18-1249. URL https://www.aclweb.org/anthology/P18-1249/.

Tassilo Klein and Moin Nabi. Attention is (not) all you need for commonsense reasoning. In Korhonen et al. (2019), pages 4831–4836. ISBN 978-1-950737-48-2. URL https://www.aclweb.org/anthology/P19-1477/.

Vid Kocijan, Oana-Maria Camburu, Ana-Maria Cretu, Yordan Yordanov, Phil Blunsom, and Thomas Lukasiewicz. Wikicrem: A large unsupervised corpus for coreference resolution. In Inui et al. (2019), pages 4302–4311. ISBN 978-1-950737-90-1. doi: 10.18653/v1/D19-1439. URL https://doi.org/10.18653/v1/D19-1439.

Vid Kocijan, Ana-Maria Cretu, Oana-Maria Camburu, Yordan Yordanov, and Thomas Lukasiewicz. A surprisingly robust trick for the winograd schema challenge. In Korhonen et al. (2019), pages 4837–4842. ISBN 978-1-950737-48-2. URL https://www.aclweb.org/anthology/P19-1478/.

Anna Korhonen, David R. Traum, and Lluís Màrquez, editors. *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, 2019. Association for Computational Linguistics. ISBN 978-1-950737-48-2. URL https://www.aclweb.org/anthology/volumes/P19-1/.

Peter Lasersohn. Event-based semantics, 2012. URL https://semanticsarchive.net/Archive/jFhNWM2M/eventbasedsemantics.pdf.

Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 311–325. Springer, 2014. ISBN 978-3-319-11557-3. doi: 10.1007/978-3-319-11558-0_22. URL https://doi.org/10.1007/978-3-319-11558-0_22.

Mark Law, Alessandra Russo, and Krysia Broda. The ILASP system for learning answer set programs. www.ilasp.com, 2015.

Mark Law, Alessandra Russo, and Krysia Broda. Iterative learning of answer set programs from context dependent examples. *TPLP*, 16(5-6):834–848, 2016. doi: 10.1017/S1471068416000351. URL https://doi.org/10.1017/S1471068416000351.

Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. Representing and learning grammars in answer set programming. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages

2919–2928. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33012919. URL https://doi.org/10.1609/aaai.v33i01.33012919.

Douglas B. Lenat, Ramanathan V. Guha, Karen Pittman, Dexter Pratt, and Mary Shepherd. CYC: toward programs with common sense. *Commun. ACM*, 33(8):30–49, 1990. doi: 10.1145/79173.79176. URL https://doi.org/10.1145/79173.79176.

Hector J. Levesque. On our best behaviour. *Artif. Intell.*, 212:27–35, 2014. doi: 10.1016/j.artint.2014.03.007. URL https://doi.org/10.1016/j.artint.2014.03.007.

Hector J. Levesque. *Common Sense, the Turing Test, and the Quest for Real AI*. The MIT Press, 1st edition, 2017. ISBN 0262036045.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press, 2012. ISBN 978-1-57735-560-1. URL http://www.aaai.org/ocs/index.php/KR/KR12/paper/view/4492.

Mike Lewis and Mark Steedman. Combined distributional and logical semantics. *TACL*, 1:179–192, 2013. URL https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/93.

Vladimir Lifschitz. What is answer set programming? In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1594–1597. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL http://www.aaai.org/Library/AAAI/2008/aaai08-270.php.

Quan Liu, Hui Jiang, Zhen-Hua Ling, Xiaodan Zhu, Si Wei, and Yu Hu. Combing context and commonsense knowledge through neural networks for solving winograd schema problems. In *2017 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 27-29, 2017*. AAAI Press, 2017. URL http://aaai.org/ocs/index.php/SSS/SSS17/paper/view/15392.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 55–60. The Association for Computer Linguistics, 2014. ISBN 978-1-941643-00-6. URL https://www.aclweb.org/anthology/P14-5010/.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Efficient HPSG parsing with supertagging and cfg-filtering. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1671–1676, 2007. URL http://ijcai.org/Proceedings/07/Papers/270.pdf.

George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, 1995. doi: 10.1145/219717.219748. URL http://doi.acm.org/10.1145/219717.219748.

Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr., Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapandula Nakashole, Emmanouil A. Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-ending learning. In Bonet and Koenig (2015), pages 2302–2310. ISBN 978-1-57735-698-1. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/10049.

Arindam Mitra and Chitta Baral. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2779–2785. AAAI Press, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12345.

Leora Morgenstern and Charles L. Ortiz Jr. The winograd schema challenge: Evaluating progress in commonsense reasoning. In Bonet and Koenig (2015), pages 4024–4026. ISBN 978-1-57735-698-1. URL http://www.aaai.org/ocs/index.php/IAAI/IAAI15/paper/view/9992.

Stephen Muggleton. Inductive logic programming. *New Generation Comput.*, 8(4):295–318, 1991. doi: 10.1007/BF03037089. URL https://doi.org/10.1007/BF03037089.

Stefan Muller and Ivan A. Sag. Introduction to HPSG, 2007. URL http://hpsg.stanford.edu/LSA07/.

Terence Parsons. *Events in the Semantics of English: A study in subatomic semantics*. MIT Press, Cambridge, MA, 1990.

Haoruo Peng, Daniel Khashabi, and Dan Roth. Solving hard coreference problems. In Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar, editors, *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 809–819. The Association for Computational Linguistics, 2015. ISBN 978-1-941643-49-5. URL https://www.aclweb.org/anthology/N15-1082/.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. Language models as knowledge bases? In Inui et al. (2019), pages 2463–2473. ISBN 978-1-950737-90-1. doi: 10.18653/v1/D19-1250. URL https://doi.org/10.18653/v1/D19-1250.

C. Pollard, I.A. Sag, Center for the Study of Language, Information (U.S.), Center for the Study of Language, and Information (U.S.). *Head-Driven Phrase Structure Grammar*. Head-driven Phrase Structure Grammar. University of Chicago Press, 1994. ISBN 9780226674476. URL https://books.google.co.uk/books?id=Ftvg8Vo3QHwC.

Ashok Prakash, Arpit Sharma, Arindam Mitra, and Chitta Baral. Combining knowledge hunting and neural language models to solve the winograd schema challenge. In Korhonen et al. (2019), pages 6110–6119. ISBN 978-1-950737-48-2. URL https://www.aclweb.org/anthology/P19-1614/.

Rashmi Prasad, Nikhil Dinesh, Alan Lee, Eleni Miltsakaki, Livio Robaldo, Aravind K. Joshi, and Bonnie L. Webber. The penn discourse treebank 2.0. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008, 26 May - 1 June 2008, Marrakech, Morocco*. European Language Resources Association, 2008. URL http://www.lrec-conf.org/proceedings/lrec2008/summaries/754.html.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. Universal dependency parsing from scratch. In Daniel Zeman and Jan Hajic, editors, *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Brussels, Belgium, October 31 - November 1, 2018*, pages 160–170. Association for Computational Linguistics, 2018. ISBN 978-1-948087-82-7. URL https://www.aclweb.org/anthology/K18-2016/.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Luc De Raedt. Inductive logic programming. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning and Data Mining*, pages 648–656. Springer, 2017. ISBN 978-1-4899-7685-7. doi: 10.1007/978-1-4899-7687-1_135. URL https://doi.org/10.1007/978-1-4899-7687-1_135.

Altaf Rahman and Vincent Ng. Resolving complex cases of definite pronouns: The winograd schema challenge. In Jun'ichi Tsujii, James Henderson, and Marius Pasca, editors, *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL 2012, July 12-14, 2012, Jeju Island, Korea*, pages 777–789. ACL, 2012. ISBN 978-1-937284-43-5. URL https://www.aclweb.org/anthology/D12-1071/.

Siva Reddy. *Syntax-Mediated Semantic Parsing*. PhD thesis, University of Edinburgh, UK, 2017. URL http://sivareddy.in/papers/siva_phd_thesis.pdf.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. Transforming dependency structures to logical forms for semantic parsing. *TACL*, 4:127–140, 2016. URL https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/807.

Adam Richard-Bollans, Lucía Gómez Álvarez, and Anthony G. Cohn. The role of pragmatics in solving the winograd schema challenge. In Andrew S. Gordon, Rob Miller, and György Turán, editors, *Proceedings of the Thirteenth International Symposium on Commonsense Reasoning, COMMONSENSE 2017, London, UK, November 6-8, 2017*, volume 2052 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017. URL http://ceur-ws.org/Vol-2052/paper17.pdf.

Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 2018. Association for Computational Linguistics. ISBN 978-1-948087-84-1. URL https://www.aclweb.org/anthology/volumes/D18-1/.

Yu-Ping Ruan, Xiaodan Zhu, Zhen-Hua Ling, Zhan Shi, Quan Liu, and Si Wei. Exploring unsupervised pretraining and sentence structure modelling for winograd schema challenge. *CoRR*, abs/1904.09705, 2019. URL http://arxiv.org/abs/1904.09705.

Josef Ruppenhofer, Michael Ellsworth, Miriam R.L. Petruck, Christopher R. Johnson, and Jan Scheffczyk. *FrameNet II: Extended Theory and Practice*. International Computer Science Institute, Berkeley, California, 2006. Distributed with the FrameNet data.

Walid S. Saba. A simple machine learning method for commonsense reasoning? A short commentary on trinh & le (2018). *CoRR*, abs/1810.00521, 2018. URL http://arxiv.org/abs/1810.00521.

Ivan A. Sag and Thomas Wasow. Syntactic theory: A formal introduction, 1999.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/6399.

Peter Schüller. Tackling winograd schemas by formalizing relevance theory in knowledge graphs. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014. ISBN 978-1-57735-657-8. URL http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7958.

Arpit Sharma. Using answer set programming for commonsense reasoning in the winograd schema challenge. *TPLP*, 19(5-6):1021–1037, 2019. doi: 10.1017/S1471068419000334. URL https://doi.org/10.1017/S1471068419000334.

Arpit Sharma, Nguyen Ha Vo, Somak Aditya, and Chitta Baral. Towards addressing the winograd schema challenge - building and using a semantic parser and a knowledge hunting module. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1319–1325. AAAI Press, 2015. ISBN 978-1-57735-738-4. URL http://ijcai.org/Abstract/15/190.

Tom De Smedt and Walter Daelemans. Pattern for python. *J. Mach. Learn. Res.*, 13:2063–2067, 2012. URL http://dl.acm.org/citation.cfm?id=2343710.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. Combinatorial sketching for finite programs. In John Paul Shen and Margaret Martonosi, editors, *Proceedings of the 12th International Conference on Architectural*

Support for Programming Languages and Operating Systems, ASPLOS 2006, San Jose, CA, USA, October 21-25, 2006, pages 404–415. ACM, 2006. doi: 10.1145/1168857.1168907. URL https://doi.org/10.1145/1168857.1168907.

Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4444–4451. AAAI Press, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14972.

Paul Trichelair, Ali Emami, Adam Trischler, Kaheer Suleman, and Jackie Chi Kit Cheung. How reasonable are common-sense reasoning tasks: A case-study on the winograd schema challenge and SWAG. In Inui et al. (2019), pages 3380–3385. ISBN 978-1-950737-90-1. doi: 10.18653/v1/D19-1335. URL https://doi.org/10.18653/v1/D19-1335.

Trieu H. Trinh and Quoc V. Le. A simple method for commonsense reasoning. *CoRR*, abs/1806.02847, 2018. URL http://arxiv.org/abs/1806.02847.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

Loïc Vial, Benjamin Lecouteux, and Didier Schwab. Sense vocabulary compression through the semantic knowledge of wordnet for neural word sense disambiguation. *CoRR*, abs/1905.05677, 2019. URL http://arxiv.org/abs/1905.05677.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Tal Linzen, Grzegorz Chrupala, and Afra Alishahi, editors, *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 353–355. Association for Computational Linguistics, 2018. ISBN 978-1-948087-71-1. URL https://www.aclweb.org/anthology/W18-5446/.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Edward A. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 3261–3275, 2019. URL http://papers.nips.cc/paper/8589-superglue-a-stickier-benchmark-for-general-purpose-language-understanding-systems.

Joseph Weizenbaum. ELIZA - a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, 1966. doi: 10.1145/365153.365168. URL https://doi.org/10.1145/365153.365168.

Michael White. OpenCCG: The OpenNLP CCG library, 2016. URL http://openccg.sourceforge.net/.

Zhi-Xiu Ye, Qian Chen, Wen Wang, and Zhen-Hua Ling. Align, mask and select: A simple method for incorporating commonsense knowledge into language representation models. *CoRR*, abs/1908.06725, 2019. URL http://arxiv.org/abs/1908.06725.

Hongming Zhang, Xinran Zhao, and Yangqiu Song. Winowhy: A deep diagnosis of essential commonsense knowledge for answering winograd schema challenge. *CoRR*, abs/2005.05763, 2020. URL https://arxiv.org/abs/2005.05763.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. Broad-coverage semantic parsing as transduction. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3784–3796. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1392. URL https://doi.org/10.18653/v1/D19-1392.

Junru Zhou and Hai Zhao. Head-driven phrase structure grammar parsing on penn treebank. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2396–2408. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1230. URL https://doi.org/10.18653/v1/p19-1230.