Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

MENG INDIVIDUAL PROJECT

Machine Comprehension Using Commonsense Knowledge

Written by:
Masturah WAN MOHD AZMI
wmb14
00931914

Supervisor:
Dr. Alessandra RUSSO

Second marker:
Dr. Krysia BRODA

June 18, 2018

Abstract

Inspired by SemEval-2018 Task 11, this project investigates the effect of incorporating commonsense knowledge on the level of machine comprehension of text. Using a logic-based rather than statistical machine learning approach, we translate text and questions to an Answer Set Programming (ASP) representation and solve this to find the relevant answers.

Using the work done by Chabierski et al. (2017) [1] as a base, we enhance their translation of text and questions by conducting a critical analysis of their CCG and λ-ASP* based approach and implementing improvements. These are done in order to be able to answer questions on complex texts, such as those in the SemEval-2018 Task 11 corpus.

Assertions from ConceptNet 5 [2], a network of human knowledge, are incorporated into the system along with relevant background knowledge rules, and we analyse the effect this has on the system's comprehension by evaluating its question-answering abilities on stories from various sources in comparison to Chabierski's system.

Acknowledgements

I would like to thank Dr. Alessandra Russo for her guidance and support throughout this project. This project would not have come this far without her guidance. I would also like to express my gratitude to my family, especially to my parents, for their constant love and unwavering support.

Contents

Chapter 1

Introduction

The problem of machine comprehension is one whose origin is commonly attributed to the Turing test (1950), which a machine passes if its responses in a conversation are indistinguishable from those of a human being [3]. Building off this idea that a machine would need to be as good as, if not better than, humans at responding to questions, a definition for machine comprehension of text could be as follows: "A machine comprehends a passage of text if, for any question regarding that text that can be answered correctly by a majority of native speakers, that machine can provide a string which those speakers would agree both answers that question, and does not contain information irrelevant to that question [4]."

One dataset that gives a measure of how well a machine does this is the Stanford Question Answering Dataset (SQuAD) [5], where questions on given texts (Wikipedia articles) have answers which can be found within the text itself and systems are not given options of answers to choose between. Many research groups use this dataset and others' performance rates as a way to benchmark their systems, and in January 2018 systems built by Alibaba and Microsoft were both able to outperform a real person in the exact match (EM) metric in correctly answering the questions in the SQuAD dataset [6]. The dataset contains questions of varying difficulty, and many of the questions the highest performing systems get wrong tend to be those that require some form of commonsense knowledge, as will be discussed in Section 3.3.

1.1 Motivation

The importance of commonsense knowledge for natural language processing (NLP) has been a topic of discussion since 1960, in the context of machine translation, such that machines would need extra knowledge in order to resolve semantic ambiguities when translating from one language to another [7][8]. For story/text comprehension, where the understanding of a text is measured by answering questions on the text, it is important that a system have some sort of background/commonsense knowledge. This is because questions may test for information that is obvious to a human, but which may not necessarily be included within the text and so without a bank of commonsense knowledge, a machine will not be able to perform as well as, or better than, humans in this task.

For example, take the sentence "Ann rang the doorbell." and the question "Does Ann have the door keys?". A human would reason that she does not have the keys since she rang the bell.

A machine with no knowledge of how the world works, however, would not be able to reason in the same way and would possibly reach a 'Don't Know' outcome. This type of question falls under story comprehension, for which commonsense knowledge tailored to the situation is particularly important for reasoning in that environment, as events in the story affect the way elements relate to one another. This differs to question answering (QA) datasets like SQuAD, where commonsense knowledge (when required) tends to be of a more general nature, like knowing that Miami is in Florida (a relation that does not change no matter the situation you are in).

1.2 Objectives

This project aims to explore the use of logic-based methods in order to create a system that will be able to process a piece of text, 'understand' it and answer questions on it with the help of pre-learned commonsense knowledge. A significant reason for choosing a logic-based rather than a statistical machine learning approach to solve this task is the fact that symbolic representations allow for more abstract concepts, like temporal relation, to be represented. This is particularly useful, for example, when learning a type of commonsense knowledge known as script knowledge, which will be discussed in Section 2.3.2.

Thus, at a high level, this project's main aims are:

1. **Translating text and questions into a sufficiently expressive logic representation**
   We use the system created by Chabierski et al. [1], which combines Combinatory Categorial Grammar (CCG) and Montague-style semantics (expressed with λ-ASP calculus) to conduct semantic analysis of text and derive Answer Set Program representations. As we would like to evaluate the system's comprehension (question-answering ability) by using a selection of stories and corresponding question and answer sets from a corpus created for a SemEval-2018 task (as described in Section 2.4), we must extend the system. This is because these stories are more complex than those used to originally evaluate the system's text comprehension, as highlighted in Section 4.

2. **Building a commonsense knowledge database**
   We want to find a way of representing commonsense knowledge acquired from various sources, as there is no one source that encompasses all the commonsense knowledge that has been collected by various efforts. We focus on one or two sources for this project, building our own database to hold relevant data that can be incorporated into our system to enhance its comprehension.

3. **Using this commonsense knowledge to aid in the answering of questions on given texts**
   With a good enough logic representation of the text and a bank of commonsense knowledge, we can then tackle the issue of identifying which commonsense knowledge can be useful when, and at what point to inject it into the system to help with question-answering.

1.3 Contributions

This project extends the work done by Chabierski et al. [1] and produces a system that is able to utilise commonsense knowledge from an existing source to aid in its comprehension abilities.

Our main contributions can be summarised as follows:

- Enhancing the English-to-ASP text translation, especially with regards to the representation of time and coreference resolution (Section 5.1)
- Extending the question-answering abilities of the system by conducting an analysis of its capabilities and addressing existing issues (Section 5.2)
- Incorporating a representation and background rules for commonsense knowledge and implementing an algorithm that injects concepts tailored to the text and question the system is processing (Section 6)
- Analysing the improvements to the system's question-answering ability by testing it on stories from
  - a hand-crafted validation set,
  - the SemEval-2018 Task 11 corpus and
  - kindergarten-level reading comprehension exercises.

  (Section 7)

Chapter 2

Background

The background information for this project can be divided into several categories, namely answer set programming (ASP), which is the logic representation used in this project, natural language processing (NLP), commonsense knowledge, SemEval-2018 (Task 11), which inspired the direction of this project and the *Chabierski System*, an existing translation system that we will be extending.

2.1 Answer Set Programming (ASP)

ASP is a form of declarative programming oriented towards difficult search problems and is particularly useful in knowledge-intensive applications [9] relying on the stable model (answer set) semantics of logic programs. An *answer set program P* is a finite set of normal rules, constraints and choice rules, defined as follows [10]:

A *literal* is either an atom p or its *default negation* not p (the negation as failure of p).

A *normal rule* has the form:

$$h \leftarrow b_1, ..., b_n, not\ c_1, ..., not\ c_m$$

where h is the head of the rule, $b_1, ..., b_n, not\ c_1, ..., not\ c_m$ as a whole is the body of the rule, and all of h, b_i, c_j are atoms.

A *constraint* has the form:

$$\leftarrow b_1, ..., b_n, not\ c_1, ..., not\ c_m$$

i.e. a rule with an empty head. The effect of adding a constraint is eliminating all answer sets that both include all b_i and exclude all c_j from the answer sets of a program.

A *choice rule* has the form:

$$l\{h_1, ..., h_m\}u \leftarrow b_1, ..., b_n, not\ c_1, ..., not\ c_m$$

where the head $l\{h_1, ..., h_m\}u$ is called an aggregate, with $l, u \in \mathbb{N}$ and $0 \leq l \leq u$. All h_i for $0 \leq i \leq m$ are atoms.

A variable V in rule R is *safe* if V occurs in at least one positive literal of R. For example, X is **not** safe in the following rules

$$p(X) \leftarrow q(Y), \text{ not } r(Y).$$
$$p \leftarrow q, \text{ not } r(X).$$

2.1.1 Stable Model Semantics

To define a stable model of a normal logic program P, we must first define a minimal Herbrand model in the context of logic programs. Given P, the *Herbrand Base* of P (HB_P) is the set of all ground (variable free) atoms that can be formed from the predicates and constants that appear in P. A Herbrand interpretation of P assigns a truth value to each ground atom $a \in HB_P$.

A *Herbrand model* M of a normal logic program P is a Herbrand interpretation in which every ground instance of a rule r in P whose body is satisfied by M, $head(r)$ is also satisfied by M. A Herbrand model M of P is *minimal* if no proper subset of M is also a Herbrand model of P.

We now define the reduct P^M of P as follows: for any set of ground atoms M of normal logic program P, a reduct P^M is a logic program that can be obtained from P by

1. removing any rule whose body contains a literal not c_i where $c_i \in M$
2. removing any negative literals in the remaining rules

M is an *answer set* (stable model) of P if and only if it is the *minimal model* of P^M. The fact that normal logic programs can have one, zero or multiple stable models leads to two different notions of entailment [11], as follows:

Brave entailment An atom a is bravely entailed by P if it is true in at least one stable model of P ($P \models_b a$).

Cautious entailment A formula a is bravely entailed by P if it is true in all stable models of P ($P \models_c a$).

2.2 Natural Language Processing (NLP)

Natural language processing (NLP) is the study of mathematical and computational modelling of various aspects of language, as well as the development of a wide range of systems [12]. It involves concepts from computer science, linguistics, logic and psychology, and NLP systems include those for speech recognition, language understanding and language generation.

Almost every NLP system has

- a grammar: finite specification of a potentially infinite number of sentences, and
- an associated parser: an algorithm that analyses a sentence and assigns one or more structural descriptions to the sentence according to the grammar, if the sentence can be characterized by the grammar

Chomsky introduced a hierarchy of grammars in [13], known as the Chomsky Hierarchy, in which he describes four sets of grammars:

Type-0 Unrestricted (recursively enumerable) grammars

Type-1 Context-sensitive grammars (CSG)

Type-2 Context-free grammars (CFG)

Type-3 Finite state grammars

Of these, we are most interested in CFGs and CSGs as many NLP systems are based on CFGs and the particular grammar we will be looking at, combinatory categorial grammar, is one that is somewhere in between a CFG and a CSG.

2.2.1 Context-Free and Context-Sensitive Grammars (CFG and CSG)

A CFG, G, consists of

- a finite set of non-terminals (e.g. S: sentence (start symbol); NP: noun-phrase; VP: verb-phrase; V: verb; ADV: adverb)
- a finite set of terminals (e.g. $John$, $clowns$, $hates$, $passionately$)
- a finite set of rewrite rules of the form $A \rightarrow W$, where A is a non-terminal and W is a string of zero or more non-terminals and terminals

Syntactic Rules
$S \rightarrow NP\ VP$
$VP \rightarrow VP\ ADV$
$VP \rightarrow V\ NP$

Lexical Rules
$NP \rightarrow John$
$NP \rightarrow clowns$
$V \rightarrow hates$
$ADV \rightarrow passionately$

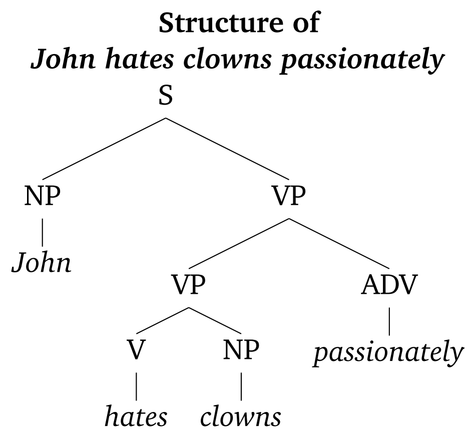


Figure 2.1: A context-free grammar

A CSG is like a CFG, except that the rewrite (syntactic) rule (on the leftmost of figure 2.1, for example) for a non-terminal is dependent on the context surrounding it. This is different to a CFG rewrite rule where the rewriting is context independent.

2.2.2 Combinatory Categorial Grammar (CCG)

CFGs are too simplistic and need to be augmented with more complex string- and tree-combining operations in order to describe various linguistic phenomena. Combinatory categorial grammar (CCG) is a grammar somewhere in between context-free and context-sensitive, known as a mildly context-sensitive grammar (MCSG). MCSG preserves many of the essential properties of CFG and is also able to capture a wide range of dependencies of language structure [12].

CCG is a form of lexicalised grammar in which the application of syntactic rules is conditioned on the category (syntactic type) of their inputs [14]. Each word is assigned to either a primitive category (like NP, noun phrase, and S, sentence) or a function category (like

$(S\backslash P)/NP$, which identifies the type and directionality of their arguments as well as the type of their result [15]).

Pure categorial grammar (CG) limits syntactic combination to rules of functional application of functions to arguments to the right or left, and this restriction limits the expressivity to the level of CFGs. CCG introduces further rules for combining categories, known as combinatory rules.

Combinatory Rules [15][14]

1. Application Rules

   The simplest of the combinatory rules are the functional application ones, as follows:

   Forward Application (>)
   $X/Y : f \quad Y : a \quad \Rightarrow \quad X : fa$

   Backward Application (<)
   $Y : a \quad X/Y : f \quad \Rightarrow \quad X : fa$

   where X and Y are syntactic categories.

   The sentence "John hates clowns", for example, yields the following derivation:

$$\dfrac{\dfrac{John}{NP : john'} \qquad \dfrac{\dfrac{hates}{(S\backslash NP)/NP : \lambda x\lambda y.hate'xy} \quad \dfrac{clowns}{NP : clowns'}}{S\backslash NP : \lambda y.hate'clowns'y}>}{S : hate'clowns'john'}<$$

2. Coordination

   The coordination rule allows constituents that are of the same category to conjoin and yield a single constituent of that category.

   Coordination (< & >)
   $X \quad conj \quad X \quad \Rightarrow \quad X$

   Take the sentence "John loathes and detests clowns":

$$\dfrac{\dfrac{John}{NP : john'} \qquad \dfrac{\dfrac{\dfrac{loathes}{(S\backslash NP)/NP : \lambda x\lambda y.loathe'xy} \quad \dfrac{and}{conj} \quad \dfrac{detests}{(S\backslash NP)/NP : \lambda x\lambda y.detest'xy}}{(S\backslash NP)/NP : \lambda x\lambda y.loathe'xy \wedge detest'xy}<\&> \quad \dfrac{clowns}{NP : clowns'}}{S\backslash NP : \lambda y.loathe'clowns'y \wedge detest'clowns'y}>}{S : loathe'clowns'john' \wedge detest'clowns'john'}<$$

3. Composition

   To allow coordination of subsequent strings (functions) that are not of the same category but where the category of one's domain matches that of the other's range, CCG allows composition on functions. There are four composition rules, as follows:

   Forward Composition ($> B$)
   $X/Y : f \quad Y/Z : g \quad \Rightarrow \quad X/Z : \lambda x.f(gx)$

   Backward Composition ($< B$)
   $Y\backslash Z : g \quad X\backslash Y : f \quad \Rightarrow \quad X\backslash Z : \lambda x.f(gx)$

   Forward Crossing Composition ($> B_{\times}$)

$X/Y : f \quad Y\backslash Z : g \quad \Rightarrow \quad X\backslash Z : \lambda x.f(gx)$

Backward Crossing Composition ($< B_\times$)

$Y/Z : g \quad X\backslash Y : f \quad \Rightarrow \quad X/Z : \lambda x.f(gx)$

where X and Y are syntactic categories.

4. Type-raising

   Type-raising rules turn arguments into functions over functions-over-such-arguments. They allow arguments to compose with the verbs that use them, and thus take part in coordinations.

   Forward Type-raising ($> T$)

   $X : a \quad \Rightarrow \quad T/(T\backslash X) : \lambda f.fa$

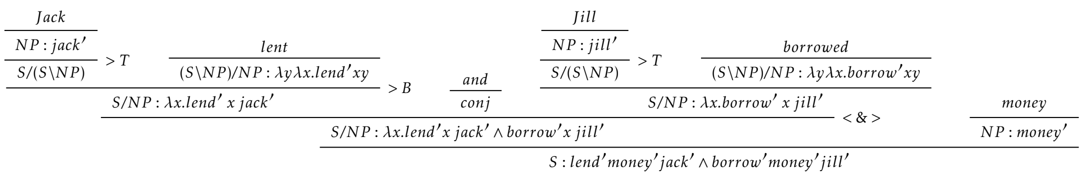
   where $X \in \{NP\}$

   Backward Type-raising ($< T$)

   $X : a \quad \Rightarrow \quad T\backslash(T/X) : \lambda f.fa$

   where $X \in \{NP, PP, AP, VP, VP', S, S'\}$

   The derivation of the sentence "Jack lent and Jill borrowed money" shows how composition and type raising work together:



We can thus create a syntactic parser for CCG, creating an algorithm that assigns one or more structural descriptions to a sentence according to the rules of CCG.

2.3 Commonsense Knowledge

Commonsense knowledge can be defined to be common knowledge about things in the world, their associations and interactions. It includes, but is not limited to, facts about events and their effects, facts about knowledge, beliefs, desires, and material objects and their properties [16]. While obvious to a human, these things would need to be represented in a system by either hard-coding or learning of some kind.

2.3.1 ConceptNet

There are many aspects of commonsense knowledge that can be added to a system, like

- Properties of elements, e.g. shiny surfaces are hard
- Taxonomy (classification), e.g. athletes are human
- Sentiment (emotions), e.g. tears mean sadness
- Structures, e.g. wheel is part of a bike

In an attempt to produce a better performing system, we thus look into inducing commonsense knowledge from ConceptNet, a dataset that has been built which covers many of these commonsense types.

ConceptNet (Liu et al., 2004)[17] (Speer et al., 2012)[18]
Currently in its fifth iteration, ConceptNet is a semantic network describing general human knowledge and how it is expressed in natural language, "designed to help computers understand the meanings of words that people use [2]". The original network was mined out of the OMCS corpus of English sentences, and this has now expanded to include data mined from other sources including:

- OMCS for other languages (Portuguese, Japanese, Dutch, Korean, French and Chinese)
- Wiktionary translations
- WordNet

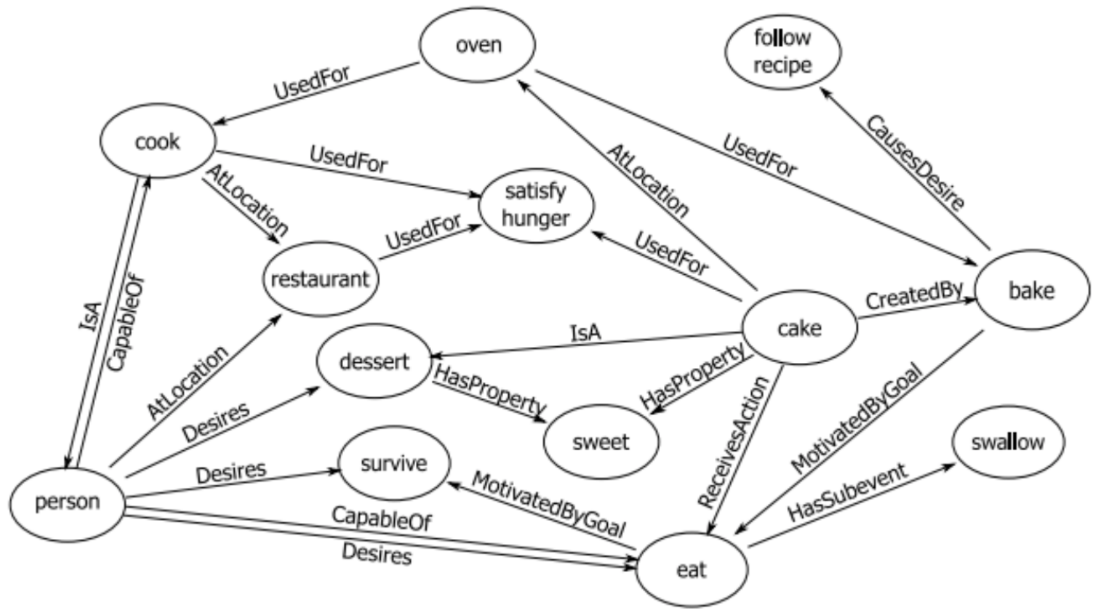


Figure 2.2: A high-level visual representation of the knowledge ConceptNet has about related concepts [18]

ConceptNet expresses *concepts* (words and phrases that can be extracted from natural text) and *assertions* of the ways these concepts relate to each other. These assertions come from the sources mentioned above (OMCS, WordNet, etc). A high-level overview can be seen in Figure 2.2. Each assertion has a label which is a predicate, like `isA` or `usedFor`. The network can be more accurately represented as a hypergraph, with the concepts as nodes and assertions as (groups of) edges between these nodes, such as in Figure 2.3. Notice the two assertions (in red and blue), which are justified by other assertions (e.g. each other), knowledge sources (e.g. Wikipedia) and processes (e.g. the ReVerb parser).

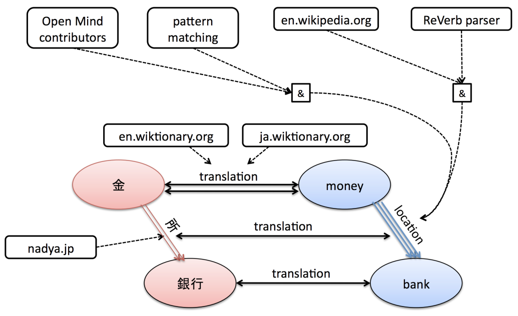


Figure 2.3: A closer look at two assertions in ConceptNet5[18]

All the relevant information about an edge is stored as properties on that edge. Each edge indicates a conjunction of sources that produced that edge, and a bundle of edges that make up an assertion is a disjunction of their conjunctions. Each conjunction is weighted with a positive or negative score, where the more positive the weight the more solidly we can conclude from this source that the assertion is true. A negative weight means that we should conclude from this source that the assertion is not true; it does **not** mean that the *negation* of the assertion is true.

Every node and edge has a Uniform Resource Identifier (URI), which contains all the information required to uniquely identify it. Each concept, for example, is represented by a URI like `/c/en/jazz/n/musical_art_form` that identifies

- that is is a concept,
- the language it is in,
- its normalised text, and
- possibly its part of speech and disambiguation.

Relations can begin with both `/c/` and `/r/`. `/r/` is used when the predicate is multilingual, e.g. `/r/isA`. For a relation taken from a sentence in a particular language, like "A bassist performs in a jazz trio" is an English sentence, a concept URI is used: `/c/en/perform_in`. An assertion URI, beginning with `/a/`, contains all the information necessary to reconstruct it. For example, "jazz is a kind of music" has the URI `/a/[/r/IsA/,/c/en/jazz/,/c/en/music/]`. We can query ConceptNet through a REST API to get all the relevant relations we require.

A more detailed explanation of how we use ConceptNet can be found in Section 6.

2.3.2 Script Knowledge

A type of commonsense knowledge is script knowledge, with a script being "a standardized sequence of events that describes some stereotypical human activity such as going to a restaurant or visiting a doctor [19]." This type of knowledge is useful for a system to have so that it is able to see how a chain of events/actions can relate to one another, and thus be able to make assumptions about temporal relations between these events based on learnt prototypical orderings. For example, if a story started with Sarah eating in a restaurant, with the help of script knowledge a system would be able to assume that this person had placed an order at an earlier point in time. Thus if the question and answer options a and b as well as the below text were given to a system with knowledge of 'going to a restaurant', it should be able to correctly choose option b as the answer.

Text: Sarah is eating in a restaurant.
Question: What did Sarah do before eating?

a. Sarah stepped in a puddle.

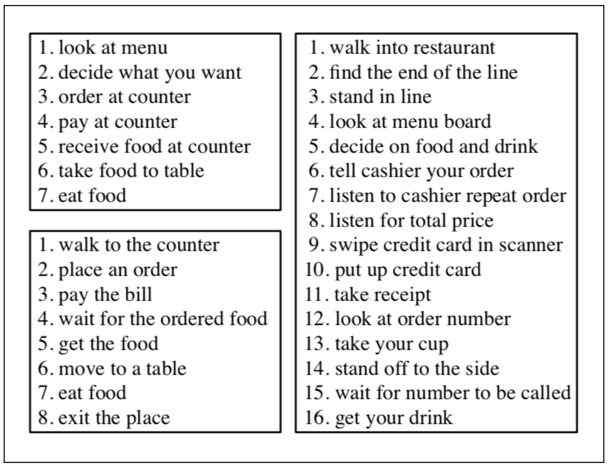
b. Sarah ordered food from a waiter.

The gathering of script knowledge has been done by various different parties and the resulting datasets from three efforts have been considered to make up part of the commonsense knowledge in our system.

1. **DeScript** corpus: a large-scale crowdsourced collection of event sequence descriptions (ESDs) (like in Figure 2.4) for various scenarios, including those that require simple general knowledge (WASHING ONE'S HAIR), more complex ones (BORROWING A BOOK FROM THE LIBRARY), ones with higher degrees of variability (GOING TO A FUNERAL) and those requiring more expert knowledge (RENOVATING A ROOM) [20]. The data was collected from 320 native English speakers describing everyday activities in small sequences of short sentences using Amazon Mechanical Turk. This was done in two phases, a pilot phase in which 10 ESDs were collected for each of 10 scenarios, and a second phase with 100 ESDs per each of the full 40 scenarios. Once the data was collected, it was manually checked to remove undesirable ESDs (i.e. those with unclear language or which misunderstood the scenario). This corpus, unlike the two below, also includes a 'gold standard' corpus of aligned event descriptions annotated by experts covering 10 scenarios with 50 ESDs each. Event descriptions (i.e. lines that make up one ESD) describing equivalent events are grouped into paraphrase sets.

2. **RKP** (by **R**egneri, **K**oller and **P**inkal) SMILE corpus: a set of ESDs for 22 scenarios of ranging complexity, for example CHILDHOOD, which is difficult to describe, MAKING SCRAMBLED EGGS, which has varied orderings of events, and WEDDING, which would differ according to culture [21]. They used Amazon Mechanical Turk to assemble data from volunteers (non-experts), getting 25 people to enter a typical sequence of events in temporal order and bullet-point style for each of the 22 scenarios. The data was manually corrected for orthography and entries with broken English or which misunderstood the scenario were discarded.

3. **Open Mind Commonsense (OMCS) Stories**: a web-collected corpus by the Open Mind Initiative [22], which contains 175 scenarios (restricted to indoor activities) with more than 40 ESDs each, the style of which resembles the RKP corpus. This was also built from information collected from the general public, but only represents a small part of the full extent of data collected through the OMCS project. More of this data is included in this project through the use of ConceptNet 5, as described in Chapter 6.



1. look at menu
2. decide what you want
3. order at counter
4. pay at counter
5. receive food at counter
6. take food to table
7. eat food

1. walk to the counter
2. place an order
3. pay the bill
4. wait for the ordered food
5. get the food
6. move to a table
7. eat food
8. exit the place

1. walk into restaurant
2. find the end of the line
3. stand in line
4. look at menu board
5. decide on food and drink
6. tell cashier your order
7. listen to cashier repeat order
8. listen for total price
9. swipe credit card in scanner
10. put up credit card
11. take receipt
12. look at order number
13. take your cup
14. stand off to the side
15. wait for number to be called
16. get your drink

Figure 2.4: Three event sequence descriptions (ESDs) for
EATING IN A FAST FOOD RESTAURANT [21]

Induction of Script Knowledge

Different approaches have been proposed for the learning of script knowledge, with three methods outlined below.

- Narrative Chains
  Chambers et al. (2008) [23] presented a new representation of structured knowledge called narrative chains, which are partially ordered sets of events centred around a common protagonist. This was extended in their 2009 work [24], which introduced typed narrative chains, which are partially ordered sets of event slots with a shared role (being a member of set of types *R*), as well as narrative schemas, which are sets of typed narrative chains modelling all actors in a set of events (as opposed to only one protagonist when you have one narrative chain). Their algorithm in (2008) uses unsupervised distributional learning with coreferring arguments as evidence of a narrative relation. Their work in (2009) uses verbs in distinct narrative chains to merge them into a single narrative schema, with shared arguments across verbs providing rich information for inducing semantic roles.

  The narrative chain schema approach (with typed chains) was a significant improvement to single narrative chains, seeing an increase in performance during narrative cloze evaluation[1], which was introduced by Chambers et al. in their (2008) paper. They found that jointly modelling arguments with events improved event clustering, and modelling related events helped argument learning: the two tasks mutually informed each other.

- Event Embedding
  Modi et al. [25] propose a statistical model to represent commonsense knowledge about prototypical event orderings. This was done by inducing distributed representations of events by composing predicate and argument representations which capture properties relevant to predicting stereotypical orderings of events from unannotated data.

[1]Narrative cloze: an event slot is removed from a narrative chain and systems are evaluated based on the position of the missing slot in the system's ranked guess list

The model they describe is limited in its present form as they chose to focus only on the ordering task, not representing all the information provided by scripts. It can, however, be extended to represent other aspects of script knowledge by modifying the learning objective.

This model was evaluated using the crowdsourced RKP corpus described above, as well as natural text in the form of news data in the Gigaword corpus. In the RKP corpus case, event embedding outperforms the F1 score of their proposed graph induction method by 13.5%. Event embedding performs well in the case of natural text too, with an accuracy improvement of 22.8% compared to the frequency-based baseline set by Chambers and Jurafsky (2008) [23].

- Event Paraphrase Sets
  Wanzare et al. (2017) [26] proposed a semi-supervised clustering approach from crowdsourced description of event sequences (ESDs). This is done by grouping individual event descriptions into paraphrase sets (representing event types) and inducing a temporal order among them. This approach exploits semantic and positional similarity, and allows for a flexible event order as opposed to the rigidity of previous approaches. Their best model (semi-supervised clustering with mixed seed data) outperforms the unsupervised model outlined in RKP in both the paraphrasing and temporal ordering tasks, as well as Modi and Titov's (2014) [25] unsupervised model for paraphrasing but does not beat it for temporal ordering.

2.4 SemEval-2018 Task 11

Task 11 of SemEval-2018[2] is entitled "Machine Comprehension using Commonsense Knowledge" and formed the initial basis of this project. Manfred Pinkal's group from Saarland University set the task, providing trial data, training and development data, and test data.

The task assesses how the inclusion of commonsense knowledge in the form of script knowledge would benefit machine comprehension systems, and systems are tested by seeing how accurately they can answer multiple choice questions based on given narrative texts about 100 different everyday activities [27].

Some of the given questions have answers that can be found directly in the text, whereas others require extra knowledge. For example, consider the following story:

```
My backyard was looking a little empty, so I decided I would plant
something. I went out and bought tree seeds. I found a spot in my
yard that looked like it would get enough sunshine. There, I dug a
hole for the seeds. Once that was done, I took my watering can and
watered the seeds.
```

Three questions that could go with this story, with the correct answers highlighted in bold, are as follows:

1. Why was the tree planted in that spot?

[2]SemEval: an international workshop on semantic evaluation intended to explore the nature of meaning in language

   A. **To get enough sunshine**
   B. There was no other space

2. What was used to dig the hole?

   A. **A shovel**
   B. Their bare hands

3. Who took the watering can?

   A. The grandmother
   B. **The gardener**

While the answer to question 1 can be found in the text, the answers to questions 2 and 3 require inferences to be made based on commonsense, namely that people tend to dig holes with a utensil rather than their bare hands (unless in special circumstances, like they don't have the right tool and are desperate or they are a child playing in the sand) and that we can call people who do gardening 'gardeners'.

2.5 The *Chabierski System* - Machine Comprehension of Text Using CCG and ASP

Chabierski et al. [1] built a system (henceforth referred to as the *Chabierski system*) that uses CCG to translate English narratives into an ASP representation, building a knowledge base that can be used, in addition to background knowledge, to answer questions on the given text. They first translate the text to an intermediate representation, λ-ASP*, which is an extension of λ-ASP (Baral et al., 2008) [28] and can handle advanced grammatical and linguistic constructions such as relativisation, control and raising. This is then translated to a general-purpose ASP representation language, designed for efficient automated learning of commonsense knowledge relevant to the given narrative. The system is also capable of translating wh- questions (who, what, where and which) that require one word answers into ASP, which would be especially useful for addressing questions in a dataset like SQuAD [5]. A detailed analysis of this system can be found in Section 4.

Chapter 3

Related Work

3.1 Story Comprehension Through Argumentation (STAR)

STAR is a system for automated narrative comprehension, implementing an argumentation-based semantics adapted to account for temporal aspects of reasoning required for story comprehension [29]. These semantics operate on a *narrative*, a set of *association rules* and *priority relations* for these rules.

The *STAR* system operates on a domain file, following and extending the Prolog syntax and comprising

- A series of **sessions** specified by the user, each representing the story narrative up to a certain scene, as well as a set of questions to be answered at that point. Sessions are of the form

  `session(s(#N),#Questions,#Visible).`, where

  - `#N` is a non-negative integer,
  - `#Questions` is a list of question names `q(#N)` to be answered by the system during this session, and
  - `#Visible` is a list of domain concepts to be shown as the system constructs its comprehension model. If `#Visible` is set to `all`, all domain concepts in the model will be shown on screen.

  These session statements give the operational definition of the session. For example, `session(s(1),[q(1),q(2)],all).` tells the *STAR* system to read up to scene `s(1)` and answer questions `q(1)` and `q(2)`.

  The narrative content of the session is given by a set of observations (scenes) of the form

  `s(#N) :: #GroundLiteral at #TimePoint.`, where

  - `#GroundLiteral` is a literal with constants replacing all arguments, and
  - `#TimePoint` is a positive integer or `always`, for literals that hold at all time points.

  An example of scenes could be:

  1. `s(0) :: person(ann) at always.`
     "ann" is always of type 'person'.

2. `s(1) :: rang(ann,doorbell) at 1.`
   ann rang the doorbell at time point 1. - extracted from a story where the first sentence is “Ann rang the doorbell.”

   The format of questions is similarly `q(#N) ?? #GroundLiteral at #TimePoint.` For example, a question for the scenes stated above could be `q(1) :: has(doorkeys,ann) at 1.`, translating to “Does Ann have the door keys?”

- The **world knowledge** (WK) used in all sessions as background knowledge for the story comprehension, accounting for the relevant commonsense knowledge required to understand that story world. We first define `#Fluents`, which are concepts that persist over and change across time. All other concepts hold only at their point of observation or inference.

  e.g. `fluents([is_a(_,_), has(_,_)]).`

  WK is then represented as associations between concepts in the form of one of three rules:

  1. Property - `p(#N) :: #Body implies #Literal.`
     at same point in time `t`
     e.g. *Residents normally have doorkeys*: `p(1) :: is_a(Person,resident) implies has(Person,doorkeys).`
  2. Causal - `c(#N) :: #Body causes #Literal.`
     where `#Body` at time `t` implies `#Literal` holds at time `t + 1`
     e.g. *Walking to the door usually gets a person close to the door*: `c(2) :: walk_to(Person,door) causes close_to(Person,door).`
  3. Preclusion - `r(#N) :: #Body precludes #Literal.`
     where `#Body` at time `t` means `#Literal` does not hold at time `t + 1`

  `p(#N)`, `c(#N)` and `r(#N)` are unique identifiers of each rule and
  `#Body = true | #Literal | #Literal,#Body`

  Priority statements of the form `#A1 >> #A2.` mean rule `#A1` takes priority over `#A2`. There are also property rules between types of statements, e.g. causal `c(#N)` rules are stronger than inertia rules (persistence of fluents) and inertia rules are stronger than property `p(#N)` rules.

With these representations, it is possible to reason about the story world at any given time, adding more information to the system during every session and learning how this new information, within the context of the given WK, affects what we thought we knew about the story world.

Looking at the corpus of STAR stories prepared by Diakidoy et al., we can see that the WK tends to be highly specialised for the story it is meant for, and is often no longer relevant when we look at other stories.

For example, in the sample story given in [29], where a character Ann is upset with her flatmate Mary and knocks on the door to her own flat to interrupt Mary from watching TV, we have specialised examples like “Walking to the door normally stops one from watching TV.” The WK in these stories represents what a human reader would typically say when asked to verbalise the knowledge they are using to comprehend a story, which makes it clear that the WK attached to each story is enough context for that story to be fully understood. This raises the question of whether the process of specialisation occurs during story comprehension or if it is stored in the human mind in its specialised contextual form. This is of interest to our project because we need to decide how to represent commonsense knowledge in our system, and how to access only the very limited set of information required for the context of each story our system tries to comprehend.

3.2 The Story Cloze Test and ROCStories

Another team working in the area of story comprehension is Mostafazadeh et al. (2016), who proposed an evaluation framework for evaluating story understanding and script learning called the Story Cloze Test [30]. This test requires a system to choose the correct ending (out of two endings) to a four-sentence story, and can be done on a corpus of 50,000 every-day life stories they built called ROCStories, which captures a rich set of causal and temporal commonsense relations between daily events.

The focus of these stories is being 'logically meaningful', such that the corpus enables a system to learn narrative structure across a range of events and can be used to train coherent story-telling models. The corpus was crowdsourced through approximate 900 workers on Amazon Mechanical Turk who were asked to write five-sentence stories, where each sentence no longer than 70 characters. They specified that stories were to have a specific beginning, ending and something happening in the middle. Each creative writer was first put through a quality control test to ensure that they understood what a good quality short story was, resulting in 49,255 of the 49,895 stories submitted being approved.

The Cloze test proved to be a challenge to all the models they tested, and Mostafazadeh et al. believe it will serve as an effective evaluation for both story understanding and script knowledge learners, which is perfect for the purposes of this project.

3.2.1 Reasoning with Heterogeneous Knowledge for Commonsense Machine Comprehension

Lin et al. (2017) [31] used the ROCStories corpus to evaluate their multi-knowledge reasoning system, in which they encoded various kinds of knowledge (including event narrative, entity semantic and sentiment coherent knowledge) as inference rules with costs. This knowledge came from mining raw text as well as relevant knowledge bases. Their reasoning model selects inference rules for a specific reasoning context using attention mechanism, and reasons by summarising all valid inference rules.

Their statistical machine learning approach significantly outperforms all baselines on the ROCStories corpus, with a 13.7% accuracy improvement on the test set. Compared to the Narrative Event Chain model (Chambers and Jurafsky, 2008) [23], their system achieves a 16.3% accuracy improvement due to the consideration of other types of commonsense knowledge (not just event narrative knowledge). They also found that it is necessary to differentiate between types of commonsense relations for machine comprehension and commonsense reasoning, as their method of modeling, distinguishing and selecting different types of commonsense relations led to a significant improvement compared to Huang et al.'s Deep Structured Semantic model (2013) [32] and Pichotta and Mooney's Recurrent Neural

Network model (2016) [33], both of which model all relations between two elements using a single semantic similarity score.

Given the success of Lin et al.'s method, it would be a good system to compare ours against to see whether our logic-based approach can outperform a statistical machine learning one.

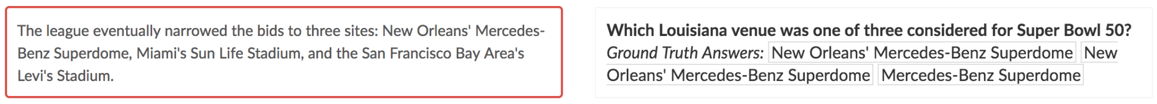
3.3 Stanford Question Answering Dataset (SQuAD)

Prior to the conception of SQuAD, reading comprehension (RC) datasets tended to have one of two limitations: they were either

1. High quality but too small to train data-intensive models (e.g. MCTest by Richardson et al., 2013 [34], with only 2640 questions), or
2. Large but did not share the same characteristics as explicit reading comprehension questions (e.g. corpus mined from 300+ thousand articles from the CNN and Daily Mail websites by Hermann et al., 2015 [35], with around 1 million Cloze [36] style[1] questions based on article summary bullet points)

SQuAD [5] was thus produced in 2016 by a group at Stanford University in order to address the need for a large, high-quality RC dataset. It consists of 107,785 questions posed by crowdworkers from the United States and Canada on a set of 23,215 paragraphs (each no shorter than 500 characters) from 536 articles randomly sampled from the top 10,000 English Wikipedia articles. The subject of these articles cover a wide range of topics from popular sporting events to abstract concepts.

When submitting questions, crowdworkers were encouraged to use their own words (i.e. not to use words in the passage). Unlike prior datasets, there is no provided selection of answers to choose from; instead, the answers to these questions are segments (*spans* of sentences) from the corresponding reading passage. Each question has three 'ground truth' answers, as seen in Figure 3.1, against which systems' answers are tested.



The league eventually narrowed the bids to three sites: New Orleans' Mercedes-Benz Superdome, Miami's Sun Life Stadium, and the San Francisco Bay Area's Levi's Stadium.

Which Louisiana venue was one of three considered for Super Bowl 50?
Ground Truth Answers: New Orleans' Mercedes-Benz Superdome | New Orleans' Mercedes-Benz Superdome | Mercedes-Benz Superdome

Figure 3.1: An example of a reading passage and corresponding question/answers from SQuAD [37]

Some world knowledge is required for a small proportion of questions in SQuAD, as can be seen from Figure 3.1 where the system must know that New Orleans is in Louisiana (and neither Miami nor San Francisco are) to answer the question correctly.

The accuracy of models attempting to answer questions in the SQuAD dataset are evaluated by two metrics, which are as follows:

1. **Exact Match** (EM), which measures the percentage of predictions that match any of the three given ground truth answers exactly

[1]Cloze procedure: a reading comprehension activity wherein words are omitted from a passage and students (or in this case systems) are required to fill in the blank

2. Macro-averaged **F1 score**, which measures the average overlap between the prediction and ground-truth answers. The maximum F1 over all three ground truth answers for each question is chosen, and then averaged over all questions.

The EM of a human performance has been exceeded by three systems at the writing of this report, first by Alibaba's *SLQA+*, then by Microsoft's *r-net+* and finally by the Joint Laboratory of HIT and iFLYTEK Research's *Hybrid AoA Reader* [37]. *r-net+* achieved the highest EM score of 82.650, 0.345 higher than the human score of 82.304. The highest F1 score of a system, however, is held by the *Hybrid AoA Reader* at 89.281, 0.788 higher than that of *r-net* but 1.94 lower than the human score of 91.221.

3.3.1 Adding Adversarial Sentences to SQuAD

Jia et al. (2017) [38] modified the SQuAD dataset by adding adversarial sentences to the paragraphs in SQuAD to test the effect of added noise on the performance of publicly available existing models, focusing on BiDAF (Seo et al., 2016) [39] and Match-LSTM (Wang and Jiang, 2016) [40]. These two are both deep learning architectures that predict a probability distribution over the correct answer [38]. They then validate their findings on the single and ensemble versions of these models on 12 more publicly available models.

They describe two concatenative adversaries, which work by leaving the original question unperturbed and adding an adversarial sentence to the end of the original paragraph. They also outline two variants on the adversaries, with all four adversaries described below:

1. ADDSENT, which uses a four-step procedure to produce human-approved grammatical sentences that look similar to the question but do not contradict the correct answer. It requires a small number of queries to the model under evaluation in order to choose the sentence that makes the model give the worst answer, and it is this sentence that is added to the end of the paragraph.

   Variant: ADDONESENT, which adds a random human-approved sentence to the paragraph.

2. ADDANY, which adds arbitrary sequences of English words to the paragraph. It does not work on all models as it assumes that the model returns a probability distribution over answers instead of a single prediction, choosing one of 20 randomised sentences that minimises the expected value of the F1 score over the models output distribution.

   Variant: ADDCOMMON, which works like ADDANY but only adds common words (the 1000 most frequent words in the Brown corpus[2])

Their results show drops in the F1 scores of all four main models (BiDAF and Match-LSTM, single and ensemble) for each of the four adversaries. ADDSENT made the average F1 score drop from 75.7% to 31.3%. Its variant, ADDONESENT, dropped the F1 score to 43.4%. ADDANY was more effective, dropping the average score down to 6.7%, but its variant ADDCOMMON dropped the average score to 46.1%.

The effects of ADDSENT and ADDONESENT were tested on the 12 extra models, and the average F1 score across all 16 models, 75.4%, dropped down to 36.4% and 46.6% respectively.

[2]A corpus of American English consisting of over 1 million words produced by Brown University [41]

However, since ADDANY and ADDCOMMON require models to return a probability distribution and not all of the other 12 models do so, these two adversaries were not run on any of the additional models.

3.4 Three-way Attention and Relational Knowledge for Commonsense Machine Comprehension

Wang et al. (2018) [42] produced the highest achieving system in SemEval-2018 Task 11 (as described in Section 2.4), using a three-way attention mechanism (Three-Way Attentive Networks *TriAN*) to model interactions between the given story text, question and answers, on top of Bi-LTSMs. Their model is pretrained on *RACE*, the largest available multiple-choice machine comprehension corpus, for ten epochs before being fine-tuned on official training data. They use a vector of GloVe embeddings to represent each word in the text, question and answers, and supplement this with additional information from part-of-speech (POS) tagging, name entity recognition, and relation embeddings based on ConceptNet to model commonsense knowledge.

These relation embeddings inspired our approach to incorporate ConceptNet into our system (described in Chapter 6), as they query ConceptNet to check whether there is an edge between words in the passage and any word in the corresponding question or answers. We used this idea of relating words in the question to words in the passage to cut down on the number of assertions we include in our logic representation, as the size of ConceptNet can potentially lead to thousands of 'relevant' concepts being added. This idea also led to our two-hop check, querying ConceptNet to get not only concepts with direct relations, but those that are related via another concept.

Chapter 4

Critical Analysis of Chabierski System

The aim of this project is to build upon the system created by Chabierski et al. [1] (henceforth referred to as the *Chabierski system*) in order to be able to 'understand' and answer questions on more complex stories. The QA functionality of the *Chabierski system* was evaluated using ten of *The (20) QA bAbI tasks* [43], whilst our main source of texts is the SemEval corpus [27].

Take the two following text and questions sets:

1. Taken from Weston et al.'s paper describing *The (20) QA bAbI tasks* [43]

   Text:

   > Mary gave the cake to Fred. Fred gave the cake to Bill. Jeff was given the milk by Bill.

   Question: Who gave the cake to Fred?

2. Taken from Story 18 of the `dev-data` corpus provided by Ostermann et al. (the organisers of SemEval-2018 Task 11) and cleaned up punctuation etc.

   Text:

   > I needed to clean up my flat. I had to get my broom and vacuum and all the cleaners I would need. I started by going around and picking up any garbage I see, like used candy wrappers and old bottles of water. I threw them away. I went around and picked up any dishes that were out and put them in the sink then washed them. I used my broom to sweep up all the dust and dirt off the hard floors, and I used the vacuum to clean up the floors that had rugs and carpets. When the hard floors were swept, I used a bucket with floor cleaner and water, and a mop, to mop them up. As the floor dried I took a rag and began dusting everything in the room. The TV, the tables and counters and everything else that was a hard surface, I used the rag to dust.

   Question: Did I use a mop?

As can be seen from the two examples above, the average sentence length and complexity of the stories in the SemEval corpus are higher than those of the bAbI dataset. Because of this, it is imperative to analyse the *Chabierski system*'s translation of both text and questions to see if improvements are required for it to be able to handle the SemEval stories.

4.1 Chabierski System Overview

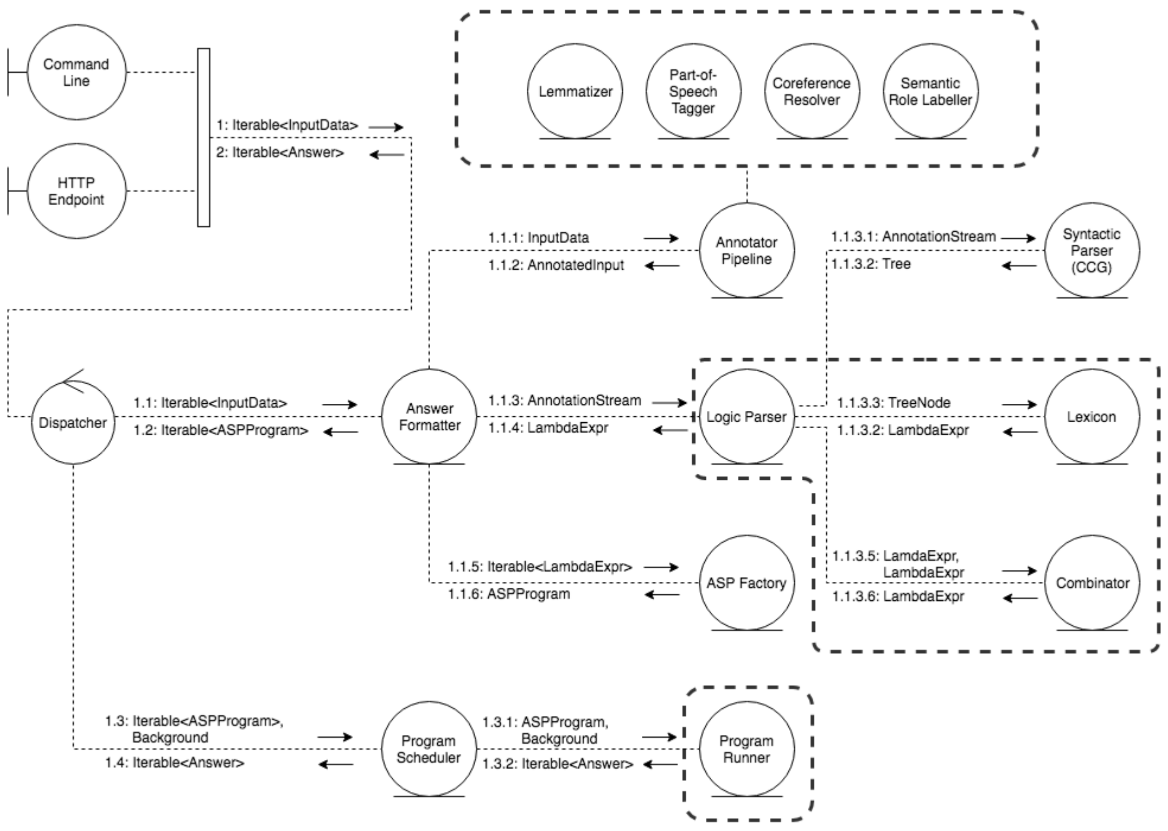


Figure 4.1: UML Diagram of Modules in the Chabierski System [44]

Figure 4.1 gives an overall view of the flow within the *Chabierski system*, showing the various modules used for annotating, parsing into logic, generating and solving an ASP program. Each of these modules are explained below.

A Input

The system takes an input (`InputData`) like the one in Figure 4.2, which includes a list of strings to represent the sentences of the text to be translated and a list of questions for the given piece of text. The 'positive' and 'negative' fields are for the system's learning mode which is not used in this project.

```
[{
    "text": [ "Mary gave the cake to Fred.",
              "Fred gave the cake to Bill." ],
    "questions": ["Who gave the cake to Fred?"],
    "positive": [],
    "negative": []
}]
```

Figure 4.2: Sample JSON Input Used by the Chabierski System

B Annotation

For each piece of `InputData`, the text and questions are annotated by two external libraries, Stanford CoreNLP [45] and EasySRL [46].

CoreNLP handles

1. Sentence splitting and tokenisation
2. Lemmatisation: grouping together inflected forms of a word and assigning the words lemma
3. Part-of-speech (POS) tagging: assigning POS tags (as defined in the Penn Treebank Project [47]) to each word, which helps to improve the robustness of λ-ASP* expression generation in a later step
4. Named entity recognition: identifying names (e.g. *Lewis Carroll*) and labelling with the appropriate class
5. Coreference resolution: determining referents of words occurring in both text and question

EasySRL is used for semantic role labelling, which assigns roles (e.g. agent, recipient, theme) to arguments of predicates. This is used to order the arguments when generating ASP predicates.

C Parsing into Logic

The CCG Parser [48] uses the previously obtained annotations to produce a (syntactic) parse tree for each sentence. This tree is then traversed to generate λ-ASP* expressions for each leaf node using the `Lexicon` module, and these are used by the `Combinator` module to compute the λ-ASP* expressions for each internal node. The `LambdaExpr` that represents the root node, which corresponds to the full sentence, is the final output.

D Generating ASP Programs and Solving for Answers

Each `LambdaExpr` is converted to `ASPPredicates` which form the final `ASPProgram`. This program is then run by Clingo [49] to generate grounded programs and thus produce answers (if any can be found).

4.2 Text Translation

An in-depth analysis of the ASP representation generated by the *Chabierski system* was conducted to evaluate the quality of the text and question translations. Chabierski et al. used online news articles as a way to evaluate the text translation of their system and found that the results showed their approach could correctly represent complex sentences [1]. This was reflected in our findings, as despite the lengthy sentences of some stories, the system was able to produce ASP representations that well reflected each sentence. That being said, there are definitely improvements that could be made to enhance their approach, as will be discussed later in this chapter. There were also issues caused by external dependencies that were more difficult to handle, and these are included in Section 4.2.1 for completeness but we do not suggest a solution for them.

4.2.1 External Dependencies

A limitation of the *Chabierski system* is its external dependencies, such as Stanford CoreNLP [45] for the majority of annotations (e.g. for coreferencing and POS tagging) and the CCG parser [48] to generate the parse tree. These tools do occasionally produce inaccurate results, which lead to inaccurate translations.

For example, to better understand the effects of the CCG parser's erroneous labels, consider the following sentence that appears in Story #208 in the SemEval-2018 Task 11 test data:

> I take my electronic list with me to the store, but I often still walk up and down most aisles just browsing for anything new or that I might want to try.

The parse tree given by the CCG parser for this sentence can be seen in Figure 4.3. An error occurs because the system tries to combine the two highlighted nodes of the tree to give the root node (first line). The types of the highlighted nodes do not match, as `S[em]\S[em]` expects to be composed with `S[em]` (an embedded declarative, e.g. a sentence beginning with a preposition like "that I might want to try"), not `S[dcl]` (a declarative sentence).

```
<T S[dcl]>
    <T S[dcl]>
        <L NP PRP I>
        <T (S[dcl]\NP)>
            <T (S[dcl]\NP)>
                <L ((S\NP)/(S\NP)) RB often>
                <T (S[dcl]\NP)>
                    <L ((S\NP)/(S\NP)) RB still>
                    <L (S[dcl]\NP) VB walk>
            <T ((S\NP)\(S\NP))>
                .
                .  rest of tree for
                .  "up and down most aisles just browsing for anything new"
                .
    <T (S[em]\S[em])>
        <L CONJ CC or>
        <T S[em]>
            <L (S[em]/S[dcl]) IN that>
            <T S[dcl]>
                <L NP PRP I>
                <T (S[dcl]\NP)>
                    <L ((S[dcl]\NP)/(S[b]\NP)) MD might>
                    <T (S[b]\NP)>
                        <L ((S[b]\NP)/(S[to]\NP)) VB want>
                        <T (S[to]\NP)>
                            <L ((S[to]\NP)/(S[b]\NP)) TO to>
                            <T (S[b]\NP)>
                                <L (S[b]\NP) VB try>
                                <L NONE . .>
```

Figure 4.3: CCG Parse Tree for
"I take my electronic list with me to the store, but I often still walk up and down most aisles just browsing for anything new or that I might want to try."

This particular issue can be solved by adding the word 'anything' to the sentence before "that I might want to try", to give the modified sentence

> I take my electronic list with me to the store, but I often still walk up and down most aisles just browsing for anything new or **anything** that I might want to try.

The CCG parser is able to produce a much better representation for this sentence than the original one. It is difficult to categorise this problem and the 'solution' found is ad-hoc, illustrating the fact that a more general solution to mitigate this problem is difficult to find. To see the effect of an incorrect CoreNLP annotation, let us consider a sentence and its parse tree and ASP representation.

Text: The dog rolls in the sand.

```
<T NP>
    <L (NP/N) DT The>
    <T N>
        <T N>
            <L (N/N) NN dog>              binaryPrep(in,c1,c2).
            <L N NNS rolls>               unaryNominal(c2,sand).
        <T (N\N)>                         unaryNominal(c1,dog).
            <T (N\N)>                     unaryNominal(c1,roll).
                <L ((N\N)/NP) IN in>      metaData(0,e0).
                <T NP>
                    <L (NP/N) DT the>
                    <L N NN sand>
            <L NONE . .>
```

Figure 4.4: CCG Parse Tree and ASP Representation for *"The dog rolls in the sand."*

The word 'rolls' is the verb in this sentence, but CoreNLP annotates it with NNS (plural noun), thus making the whole sentence a noun phrase NP rather than a sentence S. This error is carried forward into the sentence's ASP representation and the system would not be able to answer any question asked (e.g. "Who rolled in the sand?") correctly.
These inaccuracies caused by external libraries are unavoidable as there is no library that has 100% accuracy and we must thus work around this limitation by modifying the input where necessary.

4.2.2 Inaccurate Timeline

One of the major limitations of the *Chabierski system* is the way in which it handles time, which was in essence tailored for the *bAbI* dataset. Due to the nature of the 'stories' in the *bAbI* dataset, it was enough for the system to assume that the events occur in a linear ordering based on the order of the sentences in the story.

Consider Figures 4.5 and 4.6, which show sets of sentences and questions from Tasks 6 and 15 respectively of the *bAbI* dataset.

In Figure 4.5, the sentences that form the 'story' (i.e. a series of actions) are lines 1, 2, 4, 5, 7, 8, 10, 11, 13 and 14, with questions interjected to check the state of the story world at that time point. We can see that the line numbers can represent time points associated with the sentence or question on the corresponding line, thus exemplifying how some 'stories' in the dataset are linear in nature.

In Figure 4.6, the text is made up of a series of facts rather than a series of actions. With additional (learned) background knowledge to express that these are fluents which hold over time, the questions in lines 9 to 12 can be answered correctly. Therefore in this case, the ordering of the 'events' in lines 1 to 8 is not important, and can be considered linear.

```
1 Daniel went back to the kitchen.
2 Mary grabbed the apple there.
3 Is Daniel in the office?  no 1
4 Daniel journeyed to the office.
5 John went back to the office.
6 Is Daniel in the hallway?  no 4
7 Mary left the apple.
8 Daniel went to the hallway.
9 Is Daniel in the hallway?  yes 8
10 John went to the hallway.
11 Daniel picked up the milk there.
12 Is John in the kitchen?  no 10
13 John grabbed the football there.
14 Mary got the apple there.
15 Is Daniel in the hallway?  yes 8
```

Figure 4.5: Example of a Story from Task 6 of the *bAbI* dataset

```
1 Mice are afraid of wolves.
2 Cats are afraid of sheep.
3 Sheep are afraid of mice.
4 Wolves are afraid of cats.
5 Emily is a mouse.
6 Jessica is a sheep.
7 Winona is a mouse.
8 Gertrude is a mouse.
9 What is jessica afraid of? mouse 6 3
10 What is jessica afraid of? mouse 6 3
11 What is emily afraid of? wolf 5 1
12 What is emily afraid of? wolf 5 1
```

Figure 4.6: Example of a Story from Task 15 of the *bAbI* dataset

All ten tasks used to evaluate the *Chabierski system* are structured in one of the two ways shown in Figures 4.5 and 4.6, and thus for the intents and purposes of this system, time could be considered linear following the order of sentences in the story. This is why the format of the relevant predicates in the *Chabierski system* are as follows:

- **Event predicate** `ArityEvent(E,V,`X_0`,...,`X_x`)`, where
  `Arity` represents how many X_i variables there are,
  `E` is the event ID and
  `V` is a verb
- **Metadata predicate** `metaData(T,E)`, where
  `T` is a positive integer representing a time point and
  `E` is an event ID that corresponds to an `E` of an event predicate.

The `T` of the instances of the `metaData` predicate are incremented with each event (action) that occurs in the text, which makes the timeline linear. The ASP representation does not take into account the tenses of each sentence, which can instead be inferred from CoreNLP verb tags (e.g. `VBD` for **past** tense and `VBZ` for third person singular **present**). This is thus a major weakness for question answering tasks that use more complex texts which include more than one tense and where the timeline does not necessarily follow the order of sentences.

Looking at the ASP representation (Figure 4.7) of the story in Figure 4.6, we see that facts like "Mice are afraid of wolves." are not associated with an event, and thus have no time point. Sentences like "Emily is a mouse." are represented by event predicates and therefore do have time points and these are incremented as can be seen from predicates on line 17 and 20 and their corresponding metadata predicates on lines 29 and 30 respectively.

```
1 binaryModif(afraid,f0,f1).
2 binaryPrep(i0,of,afraid).
3 unaryNominal(f1,wolf).
4 unaryNominal(f0,mouse).
5 binaryModif(afraid,f2,f3).
6 binaryPrep(i1,of,afraid).
7 unaryNominal(f3,sheep).
8 unaryNominal(f2,cat).
...
17 binaryEvent(e0,be,c8,n0).
18 unaryNominal(n0,mouse).
19 unaryNominal(c8,emily).
20 binaryEvent(e1,be,c10,n1).
21 unaryNominal(n1,sheep).
22 unaryNominal(c10,jessica).
...
29 metaData(0,e0).
30 metaData(1,e1).
31 metaData(2,e2).
32 metaData(3,e3).
33 metaData(4,e4).
```

Figure 4.7: Extract from ASP Representation of Story from Figure 4.6

This representation therefore works for the purposes of the *bAbI* dataset and was a good enough method of representing time. However, for a more complex story with sentences of different tenses and questions that test this, this oversimplified representation of time is not enough. A different approach to represent time that better reflects a less straightforward story will be described in Section 5.1.1.

4.2.3 Missing Coreferences

The *Chabierski system* is able to handle coreferences to some extent, with the coreference information coming from the CoreNLP annotations, where any words that are coreferences are assigned the same `COREF` ID (as exemplified by the example below). However, some of the correct coreferences generated by CoreNLP are not included in the ASP representation because of the way IDs for nouns are being generated.

Consider the following example:

Text: My brother is a swimmer. He went to the lake.
Questions:

1. Where is he?
2. Where is my brother?
3. Where is the swimmer?

```
1 binaryEvent(e0,be,n0,n1).
2 unaryNominal(n1,swimmer).
3 binaryModif(poss,c2,n0).
4 unaryNominal(n0,brother).
5 binaryEvent(e1,go,c1,c5).
6 binaryPrep(i0,to,e1).
7 unaryNominal(c5,lake).
8 unaryNominal(c1,he).
```

Figure 4.8: ASP Representation of
My brother is a swimmer. He went to the lake.

These three questions should all give the answer 'lake', but currently only "Where is he?" gives the correct answer. This is caused largely by the missing coreferencing between 'brother', 'swimmer' and 'he'. CoreNLP is able to identify that they are all coreferences of each other, assigning each the `COREF` ID 1. This is, however, lost in translation as 'swimmer' and 'brother' are not pronouns or proper nouns and are tagged with IDs beginning with `n` (as seen in lines 2 and 4 of Figure 4.8), whereas 'he' is tagged with a `c` (`COREF`) ID as in line 8 of Figure 4.8. This means that only 'he' is attached to 'being at the lake', to which we also want 'swimmer' and 'brother' to be attached. Therefore one solution would be for all nominals which are coreferences of one another to have matching IDs, e.g.

```
unaryNominal(c1,swimmer).
unaryNominal(c1,brother).
unaryNominal(c1,he).
```

such that the predicate

```
binaryEvent(e1,go,c1,c5).
```

is associated with all three words 'swimmer', 'brother' and 'he'.

Another similar issue with the translation occurs as a result of missing coreferences for the same word (e.g. 'husband' and 'husband' in different sentences), either when

- there is an inaccuracy from CoreNLP
  While CoreNLP is able to identify coreferences quite well, it sometimes has trouble identifying that instances of the same word are referring to the same thing.
- or because of the *Chabierski system*'s approach
  As highlighted previously, it only looks for coreferences in certain scenarios and thus loses coreference information for nominals tagged with IDs beginning with different letters.

These missing coreferences result in nominal duplication, where the same word (e.g. 'husband') that refers to the same thing will have different IDs used in different predicates. For example, consider the following text, extracted from a story in the development data corpus of SemEval-2018 Task 11 [27]:

> My family and I decided that the evening was beautiful so we wanted to have a bonfire. First, my husband went to our shed and gathered some dry wood. I placed camp chairs around our fire pit. Then my husband placed the dry wood in a pyramid shape inside the fire pit.

The word 'husband' is mentioned twice, and in the corresponding ASP translation is given two different IDs (`n2` and `n5` respectively) as follows:

1. "...my husband went to our shed..."
   ```
   binaryEvent(e3,go,n2,n3).
   unaryNominal(n2,husband).
   unaryNominal(n3,shed).
   ```
2. "...my husband placed the dry wood..."
   ```
   binaryEvent(e6,place,n5,c13).
   unaryNominal(n5,husband).
   unaryNominal(c13,wood).
   ```

These two words should have the same IDs but do not, and so these different IDS are used in different (event) predicates, preventing the ASP representation from giving an overall view of all predicates related to this one word (in this case, 'husband'). This affects the ability of the system to answer questions, especially since the question translation could assign yet another ID to the word.

4.3 Question Translation

The type of questions covered by the *Chabierski system* are fairly limited, including only a subset of *wh*-questions (who, what, where) with a particular focus on one word answers and yes/no questions. The translation of these questions is done fairly well, but there are several limitations that we found when translating a variety of such questions.

4.3.1 External Dependecies - Inaccurate Annotation

As with the text translation, the system relies on external dependencies for annotations on questions. When these dependencies produce errors, the final translation of the question is also erroneous.

For example, the system uses EasySRL [46] for semantic role labelling to help order predicates, as they are sometimes ordered wrongly (e.g. the subject and object are flipped) during the logic parsing process.

Take the text and question pair

> **Text**: It is a dog.
> **Question**: What animal is it?

and the corresponding translation

```
%%% TEXT TRANSLATION %%%

binaryEvent(e0,be,c1,n0).
unaryNominal(n0,dog).
unaryNominal(c1,it).
metaData(0,e0).
metaData(1,e1).

%%% QUESTION TRANSLATION %%%

answer(e1):-
    unaryNominal(W0,animal), unaryModif(W0,X0),
    semBinaryEvent(e1,be,W0,c1), unaryNominal(c1,it).
```

Note: the `answer` predicate and `unaryModif` predicate in the question translation are both erroneous and should instead be `answer(X0)` and `unaryNominal(W0,X0)` respectively. This is a result of manual error when declaring the appropriate λ-ASP expression for this type of question where the subject is inverted (i.e. the object appears before the subject in the question).*

The arguments for a `semBinaryEvent(E,V,S,O)` predicate are:

- `E`: event ID
- `V`: verb
- `S`: subject
- `O`: object

Therefore for the given question, we would want the event predicate to represent “it is [answer]”, i.e. `semBinaryEvent(E,be,c1,W0)` with `c1` corresponding to ‘it’ and `W0` to ‘answer’. However the ordering is not correct in the obtained predicate `semBinaryEvent(e1,be,W0,c1)` which represents “answer is it”, i.e. the subject and object are inverted.

4.3.2 Exact Matching

The system generates ASP programs and finds answers by solving them, getting answers by ground predicates of type `answer(X)` in the answer set. This means that without additional information matching synonyms or like phrases together, the words in the question must exactly match those in the text for an answer to be obtained.

Consider the text and question pair

> **Text**: Mary ate an apple.
> **Question**: Did Mary consume fruit?

with translations

```
%%% TEXT %%%

binaryEvent(e0,eat,c1,n0).
unaryNominal(n0,apple).
unaryNominal(c1,mary).
metaData(0,e0).

%%% QUESTION %%%

q:-semBinaryEvent(E0,consume,c1,X0),
   unaryNominal(X0,fruit),unaryNominal(c1,mary).
answer(yes):-q.
answer(no):-not q.
```

and background knowledge

```
semBinaryEvent(E,V,S,O) :- binaryEvent(E,V,S,O).
```

We would want the answer set to include `answer(yes)` for the system to output the answer 'yes', but this is not the case because `semBinaryEvent(E0,consume,c1,X0)` and `unaryNominal(X0,fruit)` are not satisfiable in the ASP program generated by the *Chabierski system* from the given text and question.

On the contrary, the same text with the question

Question: Did Mary eat an apple?

would give the translation

```
q:-semBinaryEvent(E0,eat,c1,N0),
   unaryNominal(N0,apple),unaryNominal(c1,mary).
answer(yes):-q.
answer(no):-not q.
```

which generates the output 'yes' because `semBinaryEvent(E0,eat,c1,N0)` and `unaryNominal(N0,apple)` are satisfied with the grounding where `N0` is `n0` in the answer set generated from the text and question translation, unifying `N0` with `n0`.

This exemplifies the system's requirement that the words in the question must have an exact match in the text for an answer to be found, which is a limitation since this is not always the case with reading comprehension questions.

4.3.3 Wrong Instantiation of Variables

When translating certain text and question sets, we noticed that the question translation sometimes includes IDs that do not occur in the text translation, which means the predicates that contain those IDs as variables will never be satisfied.

Take the following as an example:

Text: My husband went to the woods. He chopped a tree.
Question: Who chopped a tree?

These translate to

```
    %%% TEXT %%%

1   binaryEvent(e0,go,n0,c3).
2   binaryPrep(i0,to,e0).
3   unaryNominal(c3,wood).
4   binaryModif(poss,c2,n0).
5   unaryNominal(n0,husband).
6   binaryEvent(e1,chop,c1,n1).
7   unaryNominal(n1,tree).
8   unaryNominal(c1,he).

    %%% TEMPORAL META DATA %%%

9   metaData(0,e0).
10  metaData(1,e1).
11  metaData(2,e2).

    %%% QUESTION %%%

12  q:-unaryNominal(W0,W1),semBinaryEvent(e2,chop,W0,N0),
13     unaryNominal(N0,tree).
14  answer(W1):-unaryNominal(W0,W1),semBinaryEvent(e2,chop,W0,N0),
15              unaryNominal(N0,tree).
```

The issue here is that the question is looking for a `chop` event that occurs at time point 2 (i.e. `semBinaryEvent(e2,chop,W0,N0)` in line 12, with the metadata in line 11), which does not occur in the text.

This is because the translation of every verb, whether in the text or question, includes an event metadata ID (i.e. the first argument in any `Event` predicate). When this ID is instantiated (always in the case of text translation but only for present or future tenses in the question translation) a corresponding `metaData` predicate and new time point are introduced. In the example above, the event ID `e2` should not have been instantiated as the question was in past tense.

The *Chabierski system* handles questions with verbs identified to be `PAST` tense by setting the first argument (event metadata ID) of the `Event` predicate to begin with a capital `E`, indicating an ungrounded variable. In the example above, however, the problem is caused by the word 'chopped' in the question being incorrectly tagged as `VBD`, a past participle (which is not identified as being a `PAST` tense verb by the system), rather than `VBN`, a verb in past tense.

These grounded ID variables tend to occur because of inaccurate annotation done by external libraries, but in cases where the resulting issue is the wrong instantiation of variables, this can be rectified by making the question predicates more general (i.e. not grounding certain variables in situations found to be prone to error). This allows for satisfiability of predicates to be determined by the matching of words rather than IDs (e.g. for a `unaryNominal(n0,book)`, we first match `book` to get its id `n0` rather than trying to match both `n0` and `book`) as well as other grounded variables which are assumed to be accurate.

4.4 Background Knowledge

The background knowledge file (Appendix A) that was used when evaluating the *Chabierski system* with the *bAbI* dataset was sufficient for those stories, but it is lacking many general rules. We recognise that the identification and addition of relevant background knowledge is a task in itself, but it should be noted that this general background knowledge file can be a starting point to be padded with rules that will complement the ASP representations generated. This project modifies and adds several such rules, to be explained further in Sections 5.1 and 5.3.

Chapter 5

Extensions Made to the *Chabierski System*

Based on our findings as explained in Section 4, we have extended the *Chabierski system* to improve the overall ASP representation obtained when inputting stories from the SemEval corpus.
As with the critical analysis, this section is broken down into three subsections: extensions made to the

1. text translation,
2. question translation and
3. background knowledge.

5.1 Text Translation

As highlighted in Section 4.2, there are two key areas in which improvements can be made, namely coreferences and time management. We further break coreferences down into two scenarios:

1. when CoreNLP annotates all the coreferences correctly but the system does not include all the coreference data and
2. when CoreNLP is unable to coreference correctly.

5.1.1 Time Management

The *Chabierski system* oversimplifies the representation of time. We therefore want to change this by including the information acquired about the tense of each `Event`, and using this to create a more informed timeline. This will allow the system to answer questions on texts with different tenses more easily.

Text: I will eat lunch later.
Question: Did I eat lunch?

```
%%% TEXT TRANSLATION %%%

unaryModif(will,e0).
unaryModif(later,e0).
binaryEvent(e0,eat,c1,f0).
unaryNominal(f0,lunch).
unaryNominal(c1,i).
metaData(0,e0).

%%% QUESTION TRANSLATION %%%

q:-semBinaryEvent(E0,eat,c1,X0),unaryNominal(X0,lunch),
    unaryNominal(c1,i).
answer(yes):-q.
answer(no):-not q.
```

Figure 5.1: Text, Question and their ASP Representation by the *Chabierski System*

For example, consider the example in Figure 5.1. With this ASP representation, the *Chabierski system* outputs the wrong answer 'yes'. This is because, due to the representation of time, everything in the text is considered to have happened. Therefore the question translation assumes that, if answering a question in the past tense, it can simply look for any event that matches, not taking into consideration that the presence of `unaryModif(will,e0)` indicates that `e0` is in the future. To counter this, we can either add rules to the background information to specify that an `Event` with modifier 'will' indicates the event has not yet happened. Alternatively, we can add tense information to the ASP representation, allowing for a more general rule stating "future events have not yet happened" to be added to the background information instead.

This is the approach we have chosen to take, and our improved system replaces the *Chabierski system*'s `metaData(T,E)` predicate with `metaData(Time, E, Tense)`, where

- `Time` is a positive integer representing a time point,
- `E` is an event ID that begins with one of `[e, p, f]` (for present, past and future respectively) and corresponds to an `E` of an event predicate, and
- `Tense` is the tense of the event, and is one of the following three values: `[past, present, future]`. Since we do include the tense in the `metaData` predicate, the `E` (event ID) does not necessarily need to begin with a letter that indicates its tense, but we chose to represent it this way to make the tense of event predicates more immediately obvious to a human reader.

With this change, we must modify the corresponding rules in the background knowledge. This change can be seen in the difference between Listing 5.1 and Listing 5.2.

Listing 5.1 Temporal Rules Used by *Chabierski System*

```
% Time points defined by the meta predicates.
previous(E1,E) :- metaData(T1,E1), metaData(T,E), T=T1+1.

% Temporal rules, semantics of 'before'.
binaryPrep(before,E0,E1) :- previous(E0,E1).
binaryPrep(before,E0,E2) :- binaryPrep(before,E1,E2), previous(E0,E1).
```

Listing 5.2 Modified Temporal Rules

```
% Time points defined by the meta predicates.
previous(E1,E2) :- metaData(T1,E1,X), metaData(T2,E2,X), T2=T1+1.
previous(E1,E2) :- metaData(T1,E1,past), metaData(T2,E2,present),
    last(E1), T2=0.
previous(E1,E2) :- metaData(T1,E1,present), metaData(T2,E2,future),
    last(E1), T2=0.

last(E) :- metaData(T,E,X), not metaData(T+1,_,X).

% Temporal rules, semantics of 'before'.
binaryPrep(before,E0,E1) :- previous(E0,E1).
binaryPrep(before,E0,E2) :- binaryPrep(before,E1,E2), previous(E0,E1).
```

With these modified rules, we get an improved timeline where all `past` predicates occur first, then the `present` and finally the `future` predicates. This timeline makes it easier to come up with temporal rules to be included in the background knowledge, and could potentially help with the implementation of further temporal questions (e.g. 'when' questions).

Consider again the example in Figure 5.1, but with this modified `metaData` predicate. Without appropriate background knowledge rules, this would still output 'yes', but now we can add a rule (line 5 in Listing 5.3) to define 'future' events as abnormal, which combined with the rule in line 2 added by Chabierski et. al makes any 'future' event predicate unsatisfiable.

```
%%% TEXT TRANSLATION %%%

unaryModif(will,f0).
unaryModif(later,f0).
binaryEvent(f0,eat,c1,f0).
unaryNominal(f0,lunch).
unaryNominal(c1,i).
metaData(0,f0,future).

%%% QUESTION TRANSLATION %%%

q:-semBinaryEvent(E0,eat,c1,X0),unaryNominal(X0,lunch),
    unaryNominal(c1,i).
answer(yes):-q.
answer(no):-not q.
```

Figure 5.2: Improved ASP Representation for Example in Figure 5.1

Listing 5.3 Semantic Rules for Future Events

```
...
semBinaryEvent(E,L,Y,Z) :- binaryEvent(E,L,Y,Z), not abBinaryEvent(E,L,Y,Z).
...
% Semantic rule for future events
abBinaryEvent(E,L,X,Y) :- metaData(_,E,future), binaryEvent(E,L,X,Y).
```

It is also worthy to note that the *Chabierski* system only assigns the future tense to sentences of type "[Noun] will [Verb Phrase]" (e.g. "I will eat lunch later"). We thus also add a rule to the source code to do indicate that sentences of form "[Noun] is going to do [Verb Phrase]" (e.g. "I am going to eat lunch later") are also future tense, which is reflected in our ASP representation.

5.1.2 Adding Missing Coreferences

As highlighted in Section 4.2.3, even when CoreNLP gives the correct annotation (i.e. correctly coreferences two nouns together), the *Chabierski system* does not always include this. We want to therefore modify the system to include all these missing coreferences, and one possible solution could be to have all coreferences assigned the same ID. Unfortunately, this is difficult to do because of the way the ASP representation is generated. The meanings of sentences are built in a bottom-up fashion using the CCG parse tree, starting with λ-ASP* expressions being assigned to words (leaf nodes) based on their annotations, and then being composed together to form the expression for parent (internal) nodes until the root node is reached and a final representation for the sentence is generated. Currently, the instantiation of parameters is not always done at its corresponding word's node, and so the system does not have access to the word's `COREF` ID. Since we are building upon an existing system, any changes made to expressions already in use and how they are built could affect its composition with other expressions, and unwanted behaviour could occur.

Due to time constraints and complexity of the problem, it was decided that a more feasible option was to implement a second traversal over the CCG parse tree and add any missing coreference information in this pass.

For example, consider the example given in Section 4.2.3, with text

> My brother is a swimmer. He went to the lake.

and the corresponding ASP representation as shown in Figure 4.8. With the second traversal in place, we get the ASP representation in Figure 5.3, where lines 5 and 6 are the new additions. With this representation, as well as the gathering of facts and generalisation of arguments (further detailed in Sections 5.1.2 and 5.2.1 respectively), we do get the correct answer for the questions "Where is the swimmer?" and "Where is my brother?", which was not the case when using the *Chabierski system*.

```
1 binaryEvent(e0,be,n0,n1).
2 unaryNominal(n1,swimmer).
3 binaryModif(poss,c2,n0).
4 unaryNominal(n0,brother).
5 unaryNominal(c1,brother).
6 unaryNominal(c1,swimmer).
7 binaryEvent(p0,go,c1,c5).
8 binaryPrep(i0,to,p0).
9 unaryNominal(c5,lake).
10 unaryNominal(c1,he).
```

Figure 5.3: New ASP Representation of
My brother is a swimmer. He went to the lake.

An effect of having the same ID represent different nominals, however, is that more answers than desired tend to fit the body of the `answer` rule. This problem is magnified by the addition of facts from ConceptNet as described in Sections 6.1 and 6.2.

Gathering Facts

When there is missing coreference information, be it because CoreNLP makes an annotation error or the system does not include all that CoreNLP gives, the system refers to different occurrences of the same noun with different IDs. To mitigate this, we have added the rules in Listing 5.4.

Listing 5.4 Rules for Gathering Facts About One Noun

```
% Collecting information for one noun
semBinaryEvent(E,V,X0,Y) :- semBinaryEvent(E,V,X1,Y),
    unaryNominal(X1,W), unaryNominal(X0,W).
semBinaryEvent(E,V,X,Y0) :- semBinaryEvent(E,V,X,Y1),
    unaryNominal(Y1,W), unaryNominal(Y0,W).
...
. and similarly for events of other arity
...
```

These two rules essentially assume that all instances of the same word are references to the same object in that story world. This is not ideal because there will of course be cases when two of the same words refer to different objects, but because of the length of the stories considered for this project, it is usually the case that this is a safe assumption to make.

These rules for gathering facts also complement the addition of the missing CoreNLP provided coreferences, as described earlier. The additions by themselves are not effective, and require these rules to help relate the new `Nominal` facts to their existing counterparts. In essence, this is the same problem as described above, where different IDs are assigned to the same reference of a noun.

Consider again the example with the ASP representation in Figure 5.3 along with the following question and its ASP representation:

Question: Where is the swimmer?

```
answer(W1):-unaryNominal(W0,W1),semBinaryEvent(e1,be,c1,W0),
    unaryNominal(c1,swimmer).
```

From the text, we have the following facts (as seen in Figure 5.3):

```
2 unaryNominal(n1,swimmer).
6 unaryNominal(c1,swimmer).
7 binaryEvent(p0,go,c1,c5).
9 unaryNominal(c5,lake).
```

Because of the rules in Listing 5.4, we also get the fact

```
semBinaryEvent(p0,go,n1,c5).
```

Thanks to the rules about the change of position of an entity defined by Chabierski et al., we also get the fact

```
semBinaryEvent(e1,be,n1,c5).
```

The combination of all these facts gives rise to `answer(lake)`, which is the correct answer. The question “Where is my brother?” for the same text requires argument generalisation in the question translation, as will be described in Section 5.2.1.

5.2 Question Translation

As highlighted by Chabierski et al. [44], the ”reliable parsing of questions using CCG grammars has been a problematic task due to smaller sizes of available training corpora.” Since the conversion from English sentences to λ-ASP* expressions relies heavily on the CCG category assigned by the parser, it is currently infeasible to expand the set of questions for which an ASP representation can be derived without substantial manual effort. This section instead focuses on making modifications to allow for more of the questions that are covered by the *Chabierski system* (i.e. who, what, where, which and yes/no questions) to be answered correctly.

Aside from small fixes, like changing the head of a λ-ASP* expression to get the right answer, the extensions made to the system with respect to the question translation are categorised into the following:

1. Argument generalisation
2. Future tense questions
3. Questions that contain pronouns that do not appear in the text
4. Multiple choice questions

5.2.1 Argument Generalisation

As described in Section 4.3.3, there are instances when the question translation sometimes includes IDs that do not occur in the text translation.

Since the example given in that section is addressed by the extension of the time management in the system (Section 5.1.1), we use another example to illustrate the way argument generalisation can help find the correct answer to a question.

Consider again the following text and ASP representation (as in Figure 5.3):

> My brother is a swimmer. He went to the lake.

```
binaryEvent(e0,be,n0,n1).
unaryNominal(n1,swimmer).
binaryModif(poss,c2,n0).
unaryNominal(n0,brother).
unaryNominal(c1,brother).
unaryNominal(c1,swimmer).
binaryEvent(p0,go,c1,c5).
binaryPrep(i0,to,p0).
unaryNominal(c5,lake).
unaryNominal(c1,he).
```

We have seen that the question "Where is he?" can be answered by the *Chabierski system* and "Where is the swimmer?" can be answered with the extension described in Sections 5.1.2 and 5.1.2.

The final question we need to be able to answer is

> Where is my brother?

```
answer(W1):-unaryNominal(W0,W1),semBinaryEvent(e1,be,n2,W0),
    binaryModif(poss,c2,n2),unaryNominal(n2,brother).
```

The ASP representation of the question uses the nominal ID `n2`, which has not been used in the text. Looking at the text translation, we can see that in order for this question to be answered, we need to unify `n2` with `n0`. We must therefore generalise `n2` to allow this unification to occur.

With this change, the question translation becomes

```
answer(W1):-unaryNominal(W0,W1),semBinaryEvent(e1,be,N0,W0),
    binaryModif(poss,c2,N0),unaryNominal(N0,brother).
```

This allows for the unification of `N0` and `n0`. Because of the rules in Listing 5.4 in Section 5.1.2 and the rules about the change of position of an entity defined by Chabierski et al., we get the facts

```
    semBinaryEvent(p0,go,n0,c5).
    semBinaryEvent(e1,be,n0,c5).
```

Again, the combination of all these facts gives rise to `answer(lake)`, which is the correct answer as required.

This generalisation of arguments is similarly done in other instances where the instantiation of variables prevented unification with existing variables in the text, and more questions are now able to be answered.

5.2.2 Future Tense Questions

Due to the *Chabierski system*'s representation of time as described in Section 4.2.2, it was not able to answer questions regarding the future (i.e. where the text contained future tense) correctly.

For example,

> **Text**: I will go to the cinema.
> **Question**: Did I go to the cinema?

outputs the answer 'yes', and

> **Question**: Will I go to the cinema?

outputs the answer no.

```
unaryModif(will,e0).
binaryEvent(e0,go,c1,c2).
binaryPrep(i0,to,e0).
unaryNominal(c2,cinema).
unaryNominal(c1,i).
metaData(0,e0).
```

Figure 5.4: "I will go to the cinema."
(*Chabierski system*)

```
q:-
  semBinaryEvent(E0,go,c1,c2),
  binaryPrep(I0,to,E0),
  unaryNominal(c2,cinema),
  unaryNominal(c1,i).
answer(yes):-q.
answer(no):-not q.
```

Figure 5.5: “Did I go to the cinema?” (*Chabierski system*)

```
metaData(1,e1).
q:-
  unaryModif(will,e1),
  semBinaryEvent(e1,go,c1,c2),
  binaryPrep(I0,to,e1),
  unaryNominal(c2,cinema),
  unaryNominal(c1,i).
answer(yes):-q.
answer(no):-not q.
```

Figure 5.6: “Will I go to the cinema?” (*Chabierski system*)

As can be seen from Figures 5.4 and 5.5, the past tense question outputs ‘yes’ because each of the predicates in the body of the question is satisfied, due to no representation of the future tense of the text as well as no background rule stating how the modified `will` affects an `Event`. Looking at the future question translation in Figure 5.6, we can see that `q` is not satisfied because `e1` cannot unify with `e0`. This could be rectified by adding a background rule about the semantics of future (will) events, but again, this is not present in the background rules for the *Chabierski system*.

Instead of adding these very specific rules about the modified ‘will’, we can use the tense information that we added to the `metaData` predicate (Section 5.1.1) and add a more general rule to the background knowledge (Listing 5.5) to specify when future events in the question should be satisfied. This is achieved because we also remove the unnecessary predicate `unaryModif(will,f1)` from the question body, relying instead on the `metaData` to give the tense of event `f1`. The predicate `binaryPrep(I0,to,f1)` is also removed because the `binaryEvent(E,go,S,L)` already represents “subject S goes to location L” and the preposition is superfluous, making it more difficult to get the correct answer.

Another advantage of this more generalised approach is that we can now also answer future questions of the form “Am I going to...?”, removing the unnecessary “going to” portion of the translation and ending up with the same question translation as “Will I go to the cinema?” (Figure 5.9). We are thus able to answer ‘going to’ questions about ‘will’ texts and vice versa, moving a step further from the exact matching (Section 4.3.2) required in the *Chabierski system*.

```
unaryModif(will,f0).
binaryEvent(f0,go,c1,c2).
binaryPrep(i0,to,f0).
unaryNominal(c2,cinema).
unaryNominal(c1,i).
metaData(0,f0,future).
```

Figure 5.7: “I will go to the cinema.” (current system)

```
q:-
  semBinaryEvent(E0,go,c1,C2),
  unaryNominal(C2,cinema).
answer(yes):-q.
answer(no):-not q.
```

Figure 5.8: “Did I go to the cinema?” (current system)

```
metaData(1,f1,future).
q:-
  semBinaryEvent(f1,go,c1,C2),
  unaryNominal(C2,cinema).
answer(yes):-q.
answer(no):-not q.
```

Figure 5.9: “Will I go to the cinema?” (current system)

Note: the `unaryNominal(c1,i)` predicate is also missing from the question bodies in Figures 5.8 and 5.9 because of their superfluous nature as well. This will be further discussed in Section 5.2.3.

Listing 5.5 Background Rule for Future Text and Questions

```
% If question and relevant text info are both future, should be
% satisfied
semBinaryEvent(E1,V,X,Y) :- metaData(T1,E1,future),
    metaData(T0,E0,future), binaryEvent(E0,V,X,Y),
    binaryPrep(before,E0,E1).
```

5.2.3 Missing Pronouns

The *Chabierski system* uses CoreNLP to annotate the text and question together so that coreferencing can occur between the two. Because of this, it is possible to have pronouns in the question that refer to something in the text without having that pronoun itself in the text.

Consider the following:

Text: Jack went to school.
Question: Where is he?

```
%%% TEXT TRANSLATION %%%

 binaryEvent(e0,go,c1,f0).
 binaryPrep(i0,to,e0).
 unaryNominal(f0,school).
 unaryNominal(c1,jack).
 metaData(0,e0).
 metaData(1,e1).

 %%% QUESTION TRANSLATION %%%

 answer(W1):-unaryNominal(W0,W1),semBinaryEvent(e1,be,c1,W0),
     unaryNominal(c1,he).
```

This ASP representation given by the *Chabierski system* can almost get the correct answer already; the only problem is that there is no `unaryNominal(c1,he).` in the text translation, and so the system cannot find the answer 'school'.

To solve this, we remove any `Nominal` predicate in the question body whose second argument is a pronoun and whose first argument is instantiated. This is because this combination means that a coreference has been found in the CoreNLP annotation and used to formulate the `Nominal` ID.

Given that the coreference information provided by CoreNLP tends to be accurate (some coreferences may be missing but what is present tends to be correct), we can simply match the coreferenced `Nominal` ID and the pronoun (e.g. 'he') does not need to be considered.

5.2.4 Queries about Actions

A question that come up often across the stories in the SemEval corpus as well as kindergarten reading comprehension (used for evaluation in Chapter 7) is of the form "What did [subject] do?"

This is not covered by the *Chabierski system,* which expects the answers required to be a noun (`Nominal`). "What did [subject] do?" instead requires an answer that is a verb, since it is looking for the actions [subject] has done.

We thus modify the question body post-predicate generation in the following way. Consider the text and question

Text: We played a game. **Question**: What did we do?

```
%%% TEXT TRANSLATION %%%

binaryEvent(p0,play,c1,n0).
unaryNominal(n0,game).
unaryNominal(c1,we).
unaryNominal(c2,game).
metaData(0,p0,past).

%%% QUESTION TRANSLATION %%%

answer(W1):-unaryNominal(W0,W1),semBinaryEvent(E0,do,c1,W0).
```

We replace the `semBinaryEvent(E0,do,c1,W0)` in the question body with `semUnaryEvent(E0,W1,c1)` and remove the `unaryNominal(W0,W1)`. This is so that we get the representation

```
answer(W1):-semUnaryEvent(E0,W1,c1).
```

This means we are looking for any action (verb) that `c1` (we) have done in the story text, thus answering the question "What did we do?"

This type of modification is only done when the 'do' `Event` predicate has an instantiated subject (argument 3), as an uninstantiated one generally indicates that the translation has an error or the question is a 'who' rather than a 'what'.

5.3 Background Knowledge

As mentioned in Section 4.4, the background knowledge file used by the *Chabierski system* on the *bAbI* dataset can be used as a starting to point to build a general background knowledge file. Along with the rules added to this file to support the changes explained in the previous sections (and the ones defined later in Chapter 6), more background knowledge has been added to the file. These additions can be found in Listing 5.6, and the whole background knowledge file in Appendix B.

Listing 5.6 Additional Background Rules

```
% Semantics of not
abUnaryEvent(E,L,Z) :- unaryEvent(E,L,Z),
    unaryModif(escnot,E).
abBinaryEvent(E,L,X,Y) :- binaryEvent(E,L,X,Y),
    unaryModif(escnot,E).
abTernaryEvent(E,L,X,Y,Z) :- ternaryEvent(E,L,X,Y,Z),
    unaryModif(escnot,E).

% Semantics of be and like (i.e. you don't stop being and liking
% something from one time point to another unless something
% affects it
binaryInitEvent(E,be,P,L) :- binaryEvent(E,be,P,L),
    not abBinaryEvent(E,be,P,L).
binaryTermEvent(E,be,P,L) :- semBinaryEvent(E,be,P,L),
    unaryModif(escnot,E).

binaryInitEvent(E,like,S,O) :- binaryEvent(E,like,S,O),
    not abBinaryEvent(E,like,S,O).
binaryTermEvent(E,like,S,O) :- semBinaryEvent(E,like,S,O),
    unaryModif(escnot,E).

% Semantics of have
binaryInitEvent(E,have,S,O) :- semBinaryEvent(E,have,S,O).
binaryTermEvent(E,have,S,O) :- semBinaryEvent(E,lose,S,O).

% Possession means I have
binaryEvent(E,have,S,O) :- binaryModif(poss,S,O),
    binaryEvent(E,_,O,_).
binaryEvent(E,have,S,O) :- binaryModif(poss,S,O),
    binaryEvent(E,_,_,O).
binaryModif(poss,S,O) :- binaryEvent(_,have,S,O).

% If an object is inside a location, it is in that location
% (the format of this is quite specific to the language of a
% specific story and could perhaps be more generalised)
binaryInitEvent(E,be,P,L) :- binaryPrep(inside,P,L),
    binaryPrep(I,in,E,P).
% If an object is placed in a location, is is in that location
binaryInitEvent(E,be,P,L) :- binaryPrep(I,in,E,L),
    binaryEvent(E,place,C,P).

% Transitivity of being in a location
binaryInitEvent(E1,be,P,L2) :- semBinaryEvent(E1,be,P,L1),
    semBinaryEvent(E2,be,L1,L2).

% Semantics of arriving and leaving
binaryInitEvent(E,be,P,L) :- semBinaryEvent(E,arrive,P,L).
binaryTermEvent(E,be,P,L) :- semBinaryEvent(E,leave,P,L).

% Generalised notion that someone's name is an identifier
% for them, e.g 'my name is X' means 'X is an identifier for I'
unaryNominal(C2, X) :- binaryEvent(E0,be,N0,C1),
    unaryNominal(C1,X), binaryModif(poss,C2,N0),
    unaryNominal(N0,name), unaryNominal(C2, _).

% Notion of "If A uses X to do Y, A uses X"
semBinaryEvent(E0,use,C1,E1) :- semTernaryEvent(E0,use,C1,E1,_).

% Notion of "If A uses X to do Y, A does Y"
semBinaryEvent(E1,V1,C1,N1) :- semTernaryEvent(E0,use,C1,N0,E1),
    semBinaryEvent(E1,V1,N0,N1).

% Notion of "If A [verb] B, A [verb]"
% e.g. "I punch a wall" ==> "I punch"
% (this may be too general a rule and should be narrowed
% down to certain verbs)
semUnaryEvent(E0,V1,C1) :- semBinaryEvent(E0,V1,C1,_).
```

5.4 Multiple Choice Questions

We are looking to evaluate the changes we make to the *Chabierski system* primarily by using the SemEval-2018 Task 11 [27] corpus, which contain stories with multiple choice questions (MCQs) of the format outlined in section 2.4. Thus we have extended the system to take another set of inputs 'answers', as can be seen in Figure 5.10. The system assumes that, if provided, one of these given answers is the correct one.

```
[{
    "text": [ "Mary gave the cake to Fred.",
              "Fred gave the cake to Bill." ],
    "questions": ["Who gave the cake to Fred?"],
    "answers": ["Mary", "Bill"],
    "positive": [],
    "negative": []
}]
```

Figure 5.10: Sample JSON Input Used by Our System

This change allows us to determine the correct user input answer based on the answer found by solving the ASP program generated from the text and corresponding question. This extra step is handled by the `AnswerVerifier` class, which can be seen in Figure 5.11, where additions to the original flow are highlighted in red.

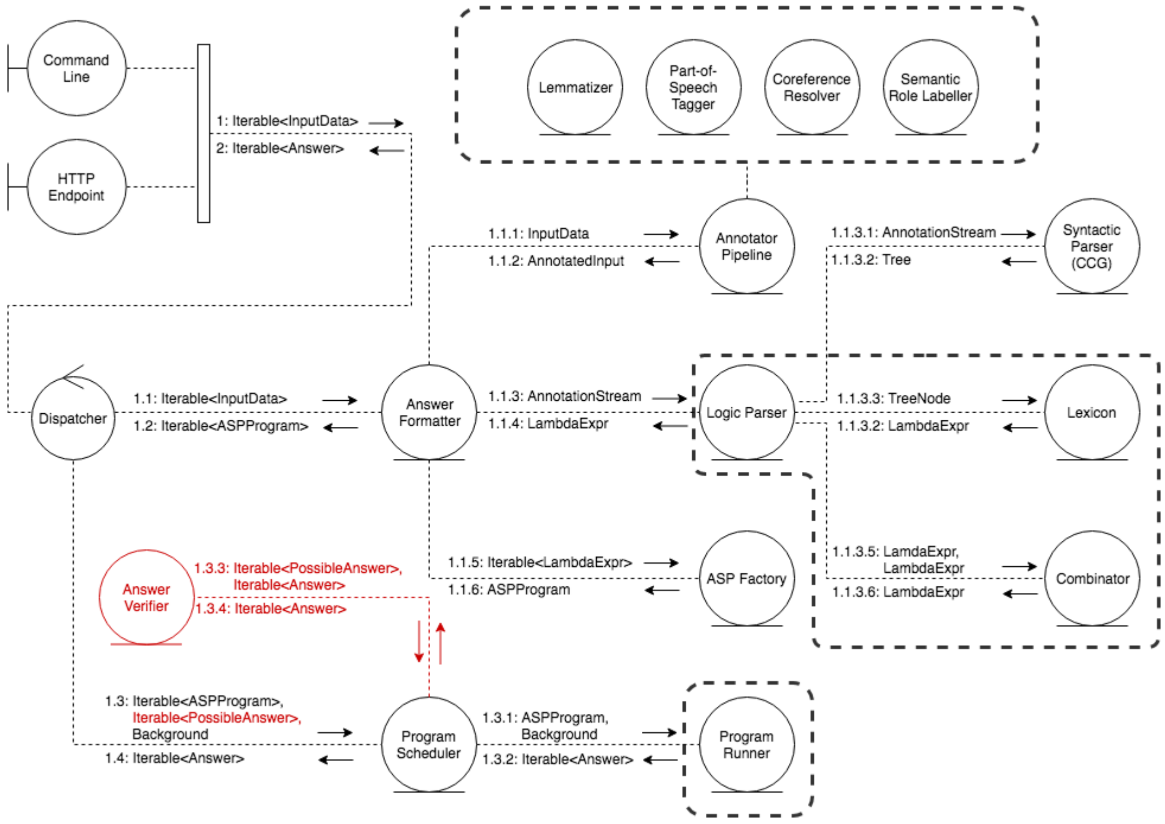


Figure 5.11: UML Diagram of Modules with the Addition of Multiple Choice

In the case where we are unable to determine the correct answer, i.e. match the found answer(s) with any given possible answer, we can use information from ConceptNet to help fill in the missing links. This will be further explained in Section 6.2.

When even the added information from ConceptNet does not yield an answer, since we make the assumption that one of the given answers is correct, we try to choose the answer with the most relevance to the text (in terms of number of occurrences in the text) as the output.

Chapter 6

ConceptNet Integration

A major limitation of the original system is the way words in the question must match words in the text exactly, otherwise an answer cannot be found. This is explained more fully in Section 4.3.2. As a way to mitigate this limitation, we have decided to supplement the information added to the ASP program by including facts fetched from ConceptNet 5 [2]. Because ConceptNet has approximately 28 million edges [50], with over 40 different relations (listed in Appendix C), we must find a way to fetch and include only the data that is most pertinent to our current text and question pair, otherwise the given ASP program would take too long to solve. We also use ConceptNet in the system's MCQ mode to try and choose the most fitting answer. The algorithms we have chosen to determine pertinent information and the best fitting answer are described in Sections 6.1 and 6.2 respectively.

Figure 6.1 shows the additional steps to the workflow of the system, with the red flow (mitigating exact matching of question to text) being covered in Section 6.1 and the blue flow in Section 6.2.

6.1 Mitigating Exact Matching

For any of the *wh*-questions, when the system returns no answer (either "`nothing`" or an empty string), we assume the problem is a mismatch of words between those used in the question and those found in the text. In the case of yes/no questions that return the answer "`no`", we also make the assumption that this is wrong to double-check that the "`no`" was not a result of mismatching. We therefore take the ASP program that was already generated and use the predicates in the body of the question to formulate queries to ConceptNet.

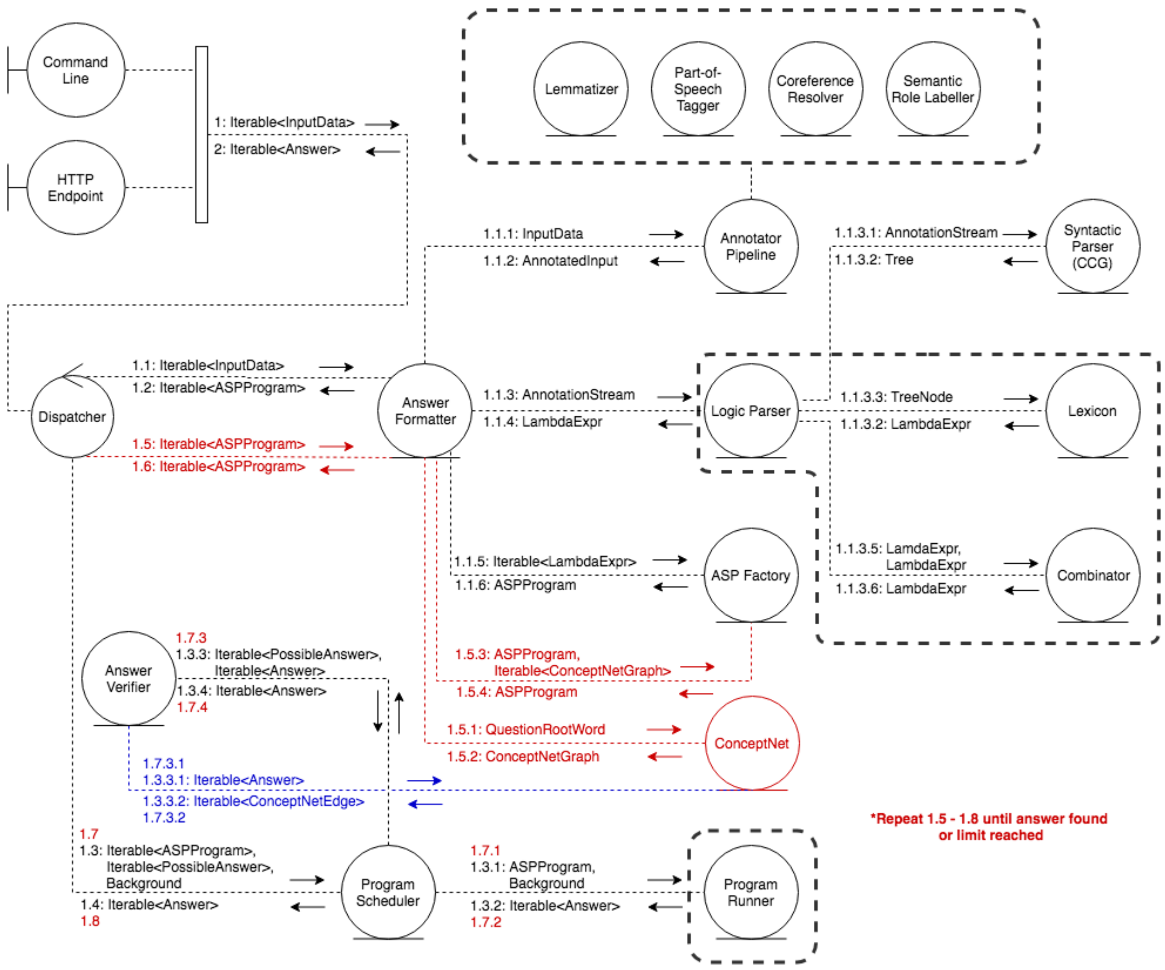


Figure 6.1: UML Diagram of Modules with the Addition of Concept Net

The red flow in Figure 6.1 illustrates the path an input takes through the system after no answer (or the answer "`no`") has been found for any of the questions given. The generated ASP program is passed back to the `AnswerFormatter`, that then uses the words in the question to form ConceptNet queries. To better understand how these queries are formatted, let us consider the following example:

Text: Mary ate an apple. **Question**: Did Mary consume fruit?

```
%%% TEXT TRANSLATION %%%

binaryEvent(p0,eat,c1,n0).
unaryNominal(n0,apple).
unaryNominal(c1,mary).
unaryNominal(c2,apple).
metaData(0,p0,past).

%%% QUESTION TRANSLATION %%%

q:-semBinaryEvent(E0,consume,c1,X0),unaryNominal(X0,fruit),
unaryNominal(c1,mary).
```

```
answer(yes):-q.
answer(no):-not q.
```

Looking at the translation above, we can see that q would never be satisfied, unless there is information relating *fruit* to *apple* and *eat* to *consume*. Thus we know we need to get relations from ConceptNet related to the *verb* and (common) *nouns* that appear in the question.

The words and phrases we query from ConceptNet (henceforth called *QuestionRootWords*) are therefore the following:

- **Verbs**, i.e. the second argument of an `Event` predicate (e.g. `consume` in `semBinaryEvent(E0,consume,c1,X0)`)
- **Nouns**, i.e. the second argument of a `unaryNominal` (e.g. `fruit` in `unaryNominal(X0,fruit)`)
  - *Note*: At this point in the translation flow, the POS tag of each word is no longer stored. Because of this, it is not possible to determine whether or not a word (e.g. `mary`) is a common or proper noun. We therefore query relations for pronouns as well as common nouns, but this is something to be improved in the future).
- **Phrases** made up of a verb + a noun (e.g. `consume_fruit` from the two examples given for verb and noun).
  - This is more pertinent for more common phrases like 'starting a fire' (`start_fire`).

Once we have these *QuestionRootWords* that we want to query, we call on the `ConceptNet` module to make the relevant API calls to formulate a `ConceptNetGraph` of related concepts. The system uses an incremental approach, in that we first check to see if adding direct relations of the *QuestionRootWords* to the ASP program will help attain an answer. Only when this does not work will we expand the graph to include two-step relations, i.e. adding all concepts related to those already in the initial graph. We currently cap the graph at two steps, stopping even if we do not find an answer.

The structure of a graph with two steps is shown in Figure 6.2, whose *QuestionRootWord* is the verb *consume*. We can see that there is a relation **MannerOf** between *consume* and *eat*, and this is added to the ASP program by the `ASPFactory` module in the form of the fact `conceptNetRelation(mannerOf,eat,consume)`.

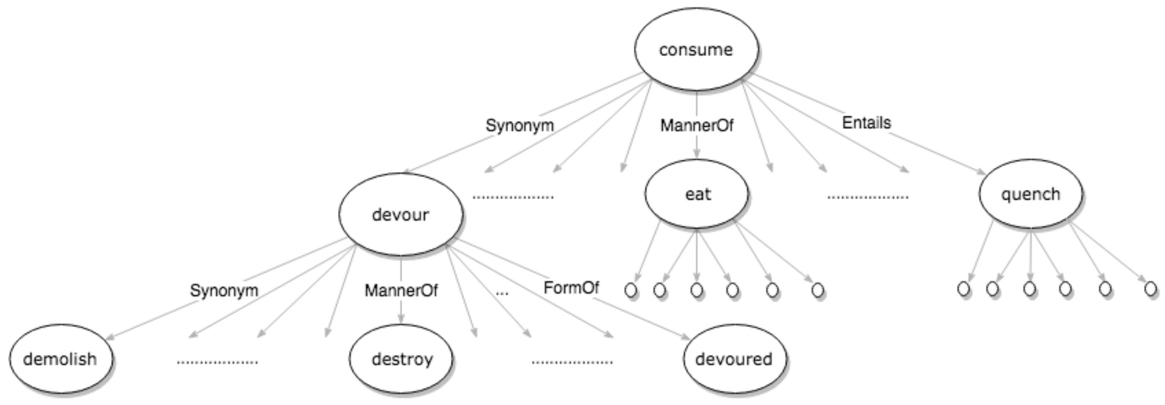


Figure 6.2: Extract of Graph Created from ConceptNet for `/c/en/consume/v`

Note that when we form a graph, we do not care about the ordering of nodes in the relation (e.g. we want relations that are both like *consume* $\xrightarrow{\text{MannerOf}}$ *spend* and *eat* $\xrightarrow{\text{MannerOf}}$ *consume*). For simplicity, the arrows in Figure 6.2 are all facing one direction, but in reality some of the arrows will be facing the opposite way.

For the relation between *consume (verb)* ↔ *eat*, it is enough to go one step. However, for *fruit (noun)* ↔ *apple*, a second step is needed because there is no relation directly linking *apple* to *fruit (noun)* in ConceptNet. Instead, we get the following:

$$\textit{apple} \xrightarrow{\text{IsA}} \textit{pome} \xrightarrow{\text{IsA}} \textit{fruit}.$$

These are added to the ASP program as the two following facts

```
conceptNetRelation(isA,apple,pome).
conceptNetRelation(isA,pome,fruit).
```

Note: ConceptNet does have a relation between *fruit* (with no noun label) and *apple*, but because we are trying to limit the size of the graph we build, we input as much information when running queries (e.g. specifying whether a word is a verb or a noun when we know it is one of the two). This is why this relation is not found when we build a graph from *QuestionRootWord fruit (noun)* and we need to go that extra step.

As we can see from the graph in Figure 6.2, there are many concepts that are related to the *QuestionRootWord* but have no relation to the text. Therefore to avoid cluttering the ASP program with unnecessary facts, we traverse the graph from the bottom up, only including concepts that are either

a. in the given text, or

b. is the parent of an included concept.

Clingo runs the ASP program after each layer of the graph is built and the relevant facts added. In the example above, it is only after the second layer is added that Clingo can solve the ASP program, with the addition of background rules in Listing 6.1, to give the answer set with `answer(yes)` and the `Dispatcher` sends 'yes' as the final answer to the question.

Listing 6.1 Background Rules for ConceptNet Relations

```
related(X,Y) :- conceptNetRelation(_,X,Y).
related(X,Y) :- conceptNetRelation(_,Y,X).
related(X,Z) :- related(X,Y), related(Y,Z).

semUnaryEvent(E,V,S) :- related(V,V1), semUnaryEvent(E,V1,S).
semBinaryEvent(E,V,S,O) :- related(V,V1), semBinaryEvent(E,V1,S,O).
semTernaryEvent(E,V,S,O,W) :- related(V,V1),
    semTernaryEvent(E,V1,S,O,W).
unaryNominal(I,W0) :- related(W0,W1), unaryNominal(I,W1).
```

The example above only demonstrates the use of queries about verbs and nouns. While we do query phrases and the relevant information is added to the ASP program, we are not currently able to use the information attained in a useful way. That is because querying phrases often returns phrases, and it is difficult to relate these in a way that can be used in the ASP program.

Take, for example,

Text: The umpire was not watching when Julie committed a foul.
Question: Who was playing tennis?

ConceptNet has the relation *play_tennis* $\xrightarrow{\text{HasSubevent}}$ *committing_foul*. We currently add `conceptNetRelation(hasSubevent, play_tennis, committing_foul)` to the ASP program, but because action phrases are essentially 'split' in the ASP representation into `Events` and `Nominals`, this added `conceptNetRelation` predicate is not actually useful. We need to find another way to relate the two phrases, which is a difficult task because the information (phrases) that come from ConceptNet does not have a set structure. This is thus work that can be done to improve the system in the future.

6.2 Choosing the Correct Answer for Multiple Choice Questions

The blue flow in Figure 6.1 shows the path taken by the system when the `AnswerVerifier` module is not able to choose one of the user input answers for a multiple choice question as the correct one based on the answers found (outputted) by running Clingo.

When this is the case, we use ConceptNet to find synonyms (relations **Synonym** and **IsA**) of the found answers. These synonyms are then used to pad the set of found answers, which is what is compared with the user input answers to choose the best fit.
Consider the example below:

Text: Mary ate an apple. **Question**: What did Mary eat? **Answers**:

A. Fruit
B. Cheese

We get the answer `apple` from the system, and so find its synonyms: [accessory fruit, apple, pome, edible fruit, fruit tree, fruit, tree, apple tree]. The system then chooses the answer 'Fruit' over 'Cheese' because the word 'fruit' appears four times in the list of synonyms, whereas the word 'cheese' does not appear at all. If an answer is not found after one query, it will then run another query for synonyms of all the previously found synonyms.

The reason we count to find a *best fit* answer rather than trying to find an exact match is because there are sometimes too many answers (not all of them correct) found with the inclusion of `conceptNetRelation` predicates and the rules in Listing 6.1, especially

```
unaryNominal(I,W0) :- related(W0,W1), unaryNominal(I,W1).
```

The longer the text and the more general the words in the question, the more facts are included from ConceptNet and the more likely Clingo will generate answer sets with words that are not always directly related. The best fit approach is to mitigate this weakness, as sometimes when the user input answers are closely related (e.g. red and pink), they can both be in the set of answers found by Clingo.

6.3 Local Database

The data in ConceptNet is available in a JSON-LD (JSON for Linked Data) API. This makes querying the data more manageable and represents the linked nature of the data in ConceptNet in a format that is easy to understand and work with.

This API returns query responses in a paginated format, meaning each JSON payload contains a limited number of 'edges' (50) and a link (value of `"nextPage"` in Figure 6.3) to retrieve the next 'page' of responses.

```
{
  "@context": [
    "http://api.conceptnet.io/ld/conceptnet5.6/context.ld.json"
  ],
  "@id": "/query?node=/c/en/start/v",
  "edges": [
    ...
    .
    . list of edges
    .
    ...
  ],
  "view": {
    "@id": "/query?node=/c/en/start/v&offset=0&limit=50",
    "@type": "PartialCollectionView",
    "comment": "There are more results. Follow the 'nextPage' link for more.",
    "firstPage": "/query?node=/c/en/start/v&offset=0&limit=50",
    "nextPage": "/query?node=/c/en/start/v&offset=50&limit=50",
    "paginatedProperty": "edges"
  }
}
```

Figure 6.3: Extract from JSON Payload Received when Querying `http://api.conceptnet.io/query?node=/c/en/start/v''`

This is a good way of managing the size of the payload, since some queries will have very large responses, but it also means that to get all the data, we need to make multiple API calls. In some cases, the number of edges to be returned rises to the thousands and the API calls required also increases accordingly.

Therefore, in order to offset the number of API calls made to the ConceptNet server, we decided to build a local database that stores any information queried using the web API. With this database in place, we only need to make each API call (along with all the subsequent calls for the remaining pages) once, and can simply query the database instead the following times we need the same information.

Our database is only made up of two tables, a `concept_net_edge` table and a `concept_net_node` table. Each edge and node JSON object in the response payload contains more information than we require, and so we store only the data that is useful (or potentially useful) for our system. These are illustrated in Figure 6.4.

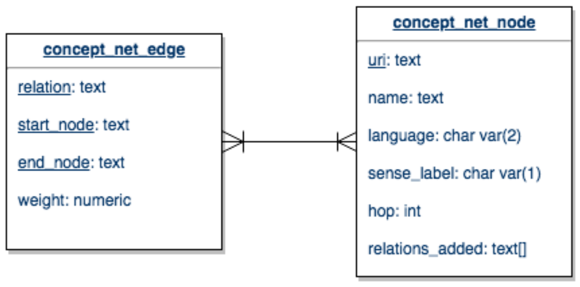


Figure 6.4: Entity-Relation Diagram of Local ConceptNet Database

A `concept_net_edge` represents a relation in ConceptNet (e.g. *apple* $\xrightarrow{\text{IsA}}$ *fruit*), and the columns of the corresponding table reflect this. The `weight` column is not currently used in our system, but is a float that represents the strength of the assertion the edge makes and could potentially be useful in the future to cut down the amount of information added to the ASP program.

The columns in the `concept_net_node` table are as follows:

- **uri**: the primary key of the table. It is Concept Net's unique URI for the node, and in essence contains the information found in the `name`, `language` and `sense` columns.
- **name**: the word the node represents.
- **language**: the language the word is in. The system currently only caters for English, but if more languages should be added in the future, this information would be useful.
- **sense_label**: `n` or `v`, depending on whether the word is a verb or a noun. Not every node has a `sense_label`.
- **hop**: an integer value to indicate the depth to which a graph has been built for this node and the relevant edges stored in `concept_net_edge`.
- **relations_added**: a list of relations that have been queried and stored in `concept_net_edge`.

We recognise that a better database design would include a `relation` entity and a junction table instead of a list of `relations_added` in the `concept_net_node` table, but our design allows for a simpler query when checking whether a certain `node` has the complete data for a specific `relation` type stored in the `concept_net_edge` table. For instance, when we only want to find as we check for either `hop > 0` or `relations_added` contains the desired `relation`.

Chapter 7

Evaluation

Our system extends the work done by Chaberski et al. (2017) [1], who presented a novel approach for translating text and questions to an ASP representation. We thus evaluate our system by assessing its ability to answer questions and compare its performance to the *Chabierski system*. We do not evaluate it quantitatively because of limitations the CCG parser imposes on our question translating capabilities (as briefly described in Section 5.2). This limits the range of questions we are able to cover, and therefore it is difficult to compare our performance against other systems, like those by participants of SemEval-2018 Task 11 who largely employed LTSM networks and other statistical machine learning methods and models.

We qualitatively evaluate our system based on its performance on answering questions from three sources:

1. **Validation Data**: hand-crafted by us to highlight the capabilities of our system
   A set of short narratives paired with questions to highlight a feature of the system were hand-crafted by us. These are similar to the examples given for each extension discussed in Chapter 5.

2. **SemEval-2018 Task 11 Corpus (MCScript)**: the manually compiled corpus provided by the organisers of Task 11
   Ten stories from the corpus, each covering a different everyday activity, along with their corresponding questions were considered. Many questions were deemed unsuitable for the system due to

   - the type of question not being covered (i.e. not a who/what/where/which or yes/no question) or
   - the question requiring commonsense information of a type we have not yet included (e.g. script knowledge).

   The questions that we did consider are similar or produce similar problems to those highlighted in Section 7.2. It is possible to hand-craft questions for each of these stories to further evaluate the system, but the questions already in the corpus provide an insight into how much the system needs to be extended in order to fully answer questions in general text comprehension (that may or may not require commonsense knowledge).

3. **Kindergarten-Level Reading Comprehension**: exercises created for students around the age of 5

Four exercises of similar style (short reading passages paired with simple MCQ questions) are used to evaluate the system's ability to cope with straightforward questions, as well as identify its weaknesses in answering them.

A sample of these with comments on the system outputs for both our system and the *Chabierski system* can be found below. For the stories from SemEval and kindergarten exercises, we change the viewpoint of the questions to match that of the story they correspond to in order for our system to be able to find any answers. The story texts are also the modified versions (i.e. we have removed or modified any problematic components like certain words or symbols). The original version of each of these texts can be found in Appendices D and E.

It should be noted that the algorithm implemented for choosing an answer out of the MCQ answer options when no answer is outputted by Clingo does not truly reflect the capabilities of our system, and any answer chosen by this *last-resort* method will be indicated.

7.1 Validation Data

This section highlights some of the capabilities of our system and the improvements we have made from the *Chabierski system*.

Story 1

My mother went to the mall. She bought a necklace.

1. What did my mother purchase?
   Chabierski: nothing
   Our system: nothing

Neither the *Chabierski* nor our system is able to answer this question because CoreNLP annotates 'purchase' in the question with the POS tag `NN` (noun) rather than with a verb tag. This leads to an erroneous ASP representation that does not have a reasonable meaning.

2. Did my mother get jewellery?
   Chabierski: nothing
   Our system: yes

As an improvement on the *Chabierski system*, our system is able to relate *get* ↔ *buy* as a direct relation from ConceptNet and *jewellery* ↔ *necklace* via *jewelry*, i.e. *necklace* $\xrightarrow{\text{IsA}}$ *jewelry* $\xrightarrow{\text{FormOf}}$ *jewellery*. We are also able to unify 'my mother' with 'she', a coreference that is missing in the translation given by the *Chabierski system*.

Story 2

I ate an apple because I was hungry.
I then used soap to wash my hands.

1. Did I consume fruit?
   Chabierski: no
   Our system: yes

We are able to answer this question correctly because we were able to relate *consume* ↔ *eat* and *fruit* ↔ *apple*, thanks to assertions fetched from CoreNLP, namely *consume* $\xrightarrow{\text{Synonym}}$ *eat* and *apple* $\xrightarrow{\text{IsA}}$ *pome* $\xrightarrow{\text{IsA}}$ *fruit*.

2. Did I wash my hands?
   Chabierski: no
   Our system: yes

A background rule added to our system says that "If [A] uses [B] to do [C], then [A] does [C]." This is a rule that could perhaps be generalised further (i.e. be true for more than the verb 'use'), but it is an example of a fairly general rule that enhances the knowledge of the system.

3. Did I use soap?
   Chabierski: yes
   Our system: yes

4. Was I hungry?
   Chabierski: yes
   Our system: yes

These two questions generate the correct result from both systems as no background rules or extra knowledge are required.

Story 3

I called her. I will call him.

#	Question	Correct	Our Output	Chabierski Output
1	Have I called her?	yes	yes	no
2	Have I called him?	no	no	no
3	Will I call her?	no	no	no
4	Will I call him?	yes	yes	no
5	Am I going to call her?	no	no	no
6	Am I going to call him?	yes	yes	no
7	Did I call her?	yes	yes	yes
8	Did I call him?	no	no	yes

This example highlights the ability of our system to accurately discern what events have happened and which have not. Question 1 was wrongly answered by the *Chabierski system* due to the wrong instantiation of the event ID variable, which we generalised in our system. The *Chabierski system* also does not have a good representation of the future, which is why questions 3 - 6 are wrong. Question 8 is also incorrect because all text events are considered to have happened before the question time point, and so “I call him” (despite the ‘will’ modifier) is still considered to have happened.

Our system gives the same correct answers for these 8 questions but with the text “I (have) called her. I am going to call him.” Modifications need to be made to the source code to recognise other phrases as future tense (e.g. “I am about to...”) but this example shows the effectiveness of our time management system coupled with our added background rules, and our ability not to require exact matching between text and question.

7.2 SemEval-2018 Task 11 Corpus

We look at three different narratives and their accompanying set of questions and answers. Many of the questions are not covered by our system, and some of them require modifications in order for our system to produce an answer.

The way answers are represented are as follows:

- The correct answer for each question is in **bold**
- The answer chosen by our system is marked with an arrow (←)
  - A *last-resort* is added next to the arrow if the MCQ last-resort method is the one that chooses that answer
- The answers outputted by Clingo for the *Chabierski system* is given below the question

Story 1: Ordering a Pizza

One afternoon when I was visiting with friends, we suddenly got hungry. So, we decided to order a pizza. I looked in the telephone book and found several pizza restaurants that would deliver. We all talked it over and settled on ordering a pizza from Dominos. Then we discussed what we would like on our pizza and what size we should get. I called the number and gave our order to the person who answered. We ordered a large pepperoni and onion pizza with extra cheese. He told me that our pizza would cost fifteen dollars. I said that would be fine and asked if the pizza could be delivered. He confirmed it could be and asked for my address. I then gave him my name, address, and phone number. He told me that our pizza would be delivered in thirty minutes. When the delivery boy arrived, we paid for our pizza and included a two dollar tip. Then we enjoyed eating our pizza.

1. Who ordered the pizza?
   Chabierski: nothing

   A. **The narrator ←**

   B. The dog

We are able to answer this question thanks to argument generalisation and collection of facts, as the identifier for ‘pizza’ in the question is not the same as the one in the sentence “We ordered a large pepperoni pizza...”

2. Did we order any other entrees?
   Chabierski: no

   A. **No** ←

   B. Yes

Both systems give the correct answer but the answer given by our system is 'more correct' because it tries to look for entrees (as defined by ConceptNet) and does not find any.

3. Where did they order it?
   Chabierski: nothing

   A. Pizza Hut ← *last-resort*

   B. **from Dominos**

The translation of the question by the *Chabierski system* is

```
answer(W1):-
    unaryNominal(W0,W1),
    semBinaryEvent(E0,
    order,c59,c7),
    unaryNominal(c7,it),
    unaryNominal(c59,they).
```

More manual work is required to improve the translation of this question. The traversal of the CCG parse tree encounters an error because it runs out of parameters: it does not know where to get the answer variable from because 'did they order it?' is represented as `semBinaryEvent(E0,order,c59,c7)` ("they ordered it"), with no (nominal) variable to unify with the ID `W0` of answer predicate `unaryNominal(W0,W1)`.

This error is reflected in our system as well, and it thus uses the *last-resort* method to choose an answer. It chooses 'Pizza Hut' over 'Dominos' because 'Dominos' only appears once in the story, whereas 'pizza' appears multiple times and the algorithm does a count of each word that appears in the answer (not including words like 'a', 'the', 'some').

Story 2: Making a Bonfire

My family and I decided that the evening was beautiful so we wanted to have a bonfire. First, my husband went to our shed and gathered some dry wood. I placed camp chairs around our fire pit. Then my husband placed the dry wood in a pyramid shape inside the fire pit. He then set some small kindling ablaze. Once the kindling reached the dry wood, it set the wood on fire. We then sat around the fire for some time, adding more logs as the previous ones burned out. We cooked marshmallows on sticks over the open flames. When the marshmallows were golden brown, we placed them between two graham crackers with chocolate pieces. We ate our smores as we joked, laughed and told stories around our beautiful fire. When we finished, I put away our camp chairs. My husband made sure the fire was out by dousing it with some water and we went inside to bed.

1. Who started the fire?
   Chabierski: nothing
   A. Wife
   B. **Their husband lit it** ← *last-resort*

2. Who set the kindling ablaze?
   Chabierski: he
   A. Wife
   B. **Their husband lit it** ←

The question in the SemEval corpus is Question 1, but we are unable to answer it even with ConceptNet because we do not have a way to relate *start fire* with *set kindling ablaze* in a way that will help Clingo solve for the answer, even though these connections do vaguely exist in ConceptNet.

Modifying the question to "Who set the kindling ablaze?", a phrase that appears directly in the text, the *Chabierski system* gives the correct answer 'he', but is unable to coreference this with 'husband', which is the answer we desire. With the enhancements to coreferencing resolution in our system, this is fixed and we do get the answer 'husband', resulting in choosing the correct MCQ answer.

My family and I decided that the evening was beautiful so we wanted to have a bonfire. First, my husband went to our shed and gathered some dry wood. I placed camp chairs around our fire pit. Then my husband placed the dry wood in a pyramid shape inside the fire pit. He then set some small kindling ablaze. Once the kindling reached the dry wood, it set the wood on fire. We then sat around the fire for some time, adding more logs as the previous ones burned out. We cooked marshmallows on sticks over the open flames. When the marshmallows were golden brown, we placed them between two graham crackers with chocolate pieces. We ate our smores as we joked, laughed and told stories around our beautiful fire. When we finished, I put away our camp chairs. My husband made sure the fire was out by dousing it with some water and we went inside to bed.

3. Where did they make the bonfire?
   Chabierski: nothing
   A. In the garage
   B. **Their fire pit** ← *last-resort*

Similarly to Questions 1 and 2, the question in the SemEval corpus is Question 3, but we are unable to answer because we do not have a way to relate *make bonfire* with *place dry wood in pyramid shape*. This is quite a complex representation and we are unable to use the concepts gotten from ConceptNet to make a connection between the two.

4. Where is the wood?
   Chabierski: nothing
   Our system: [pyramid shape fire pit wood kindling ablaze]
   A. In the garage
   B. **Their fire pit** ←

Modifying the question and simplifying the underlying concept to “Where is the wood?”, we can find the correct answer thanks to the additional background knowledge implemented in our system about the semantics of ‘inside’ and ‘in’. Looking at the raw output from our system, `[pyramid shape fire pit wood kindling ablaze]`, we can see that there is extra information given that does not answer our question, and that is because of the background knowledge also trying to gather facts, often resulting in too many connections being made. This problem is discussed further in Section 7.5, and a way to mitigate it discussed in 8.1.

My family and I decided that the evening was beautiful so we wanted to have a bonfire. First, my husband went to our shed and gathered some dry wood. I placed camp chairs around our fire pit. Then my husband placed the dry wood in a pyramid shape inside the fire pit. He then set some small kindling ablaze. Once the kindling reached the dry wood, it set the wood on fire. We then sat around the fire for some time, adding more logs as the previous ones burned out. We cooked marshmallows on sticks over the open flames. When the marshmallows were golden brown, we placed them between two graham crackers with chocolate pieces. We ate our smores as we joked, laughed and told stories around our beautiful fire. When we finished, I put away our camp chairs. My husband made sure the fire was out by dousing it with some water and we went inside to bed.

5. What did we do at the bonfire?
   Chabierski: nothing

   A. **Talk and cook smores** ← *last-resort*

   B. Dance around

We are able to answer this question when we remove “at the bonfire” because it adds predicates that need to be satisfied which will not be satisfied because none of the sentences in the text say anything about doing things “at the bonfire”. This is thus something that can be removed from the question, but we could not formulate a general rule for when this kind of information is superfluous and so we remove it on an ad-hoc basis.

6. What did we do?
   Chabierski: nothing

   A. **Talk and cook smores** ←

   B. Dance around

Question 6 initially gives the raw output `[laugh joke finish add sit tell eat go want have be]`, all verbs that ‘we’ do in the text. We try to match this to one of the given answers, but this fails as they do not directly relate. We thus fetch synonyms of these verbs from ConceptNet to try and find a connection, and this list of synonyms helps us choose the correct final answer.

Story 3: Cleaning Up a Flat

I needed to clean up my flat. I had to get my broom and vacuum and all the cleaners I would need. I started by going around and picking up any garbage I see, like used candy wrappers and old bottles of water. I threw them away. I went around and picked up any dishes that were out and put them in the sink then washed them. I used my broom to sweep up all the dust and dirt off the hard floors, and I used the vacuum to clean up the floors that had rugs and carpets. When the hard floors were swept, I used a bucket with floor cleaner and water, and a mop, to mop them up. As the floor dried I took a rag and began dusting everything in the room. The TV, the tables and counters and everything else that was a hard surface, I used the rag to dust.

1. Did I use a mop?
   Chabierski: no

   A. No

   B. **Yes after sweeping** ←

The background knowledge added for "If [A] uses [B] to do [C], then [A] does [C]" helps our system answer this question.

2. Where did I vacuum?
   Chabierski: nothing

   A. Hardwood floor

   B. **Rooms with rugs and carpet** ← *last-resort*

The relevant sentence in the story ("I used the vacuum to clean up the floors that had rugs and carpets") is translated into a representation that is too convoluted to be able to answer without very specific background rules.

```
unaryNominal(c1,i).
ternaryEvent(p10,use,c1,c25,e5).
unaryNominal(c25,vacuum).
binaryEvent(e5,clean_up,c25,n3).
unaryNominal(c26,carpet).
unaryNominal(n4,rug).
binaryEvent(p11,have,n3,c26).
binaryEvent(p11,have,n3,n4).
unaryNominal(n3,floor).
```

In summary, the stories and questions in the SemEval corpus are complex, sometimes in ways that are not immediately obvious to humans. The ASP representation generated by the system is usually well done in that it is understandable and systematic, but this does not mean that it is easy to use to answer questions. With the help of additional background knowledge and ConceptNet, we are able to answer more than what the *Chabierski system* is capable of, but there is still a long way to go to be able to answer the questions given with no modification.

7.3 Kindergarten Reading Comprehension

Looking at the reading comprehension exercises for various grade levels, we deemed the kindergarten level to be the most suitable for evaluating our system. Higher levels start asking more complex 'how' and 'why' questions which are not yet covered by our system.

As before, the way answers are represented are as follows:

- The correct answer for each question is in **bold**
- The answer chosen by our system is marked with an arrow (←)
  - A *last-resort* is added next to the arrow if the MCQ last-resort method is the one that chooses that answer
- The answers outputted by Clingo for the *Chabierski system* is given below the question

Story 1

See my dog. We like to play outside. We run and jump in the sun.

1. What pet do I have?
   Chabierski: nothing

   A. **Dog** ←
   B. Cat
   C. Bird

The system is now able to answer this question because it can relate *pet* ↔ *dog*. There was also an error in the λ-ASP* representation of subject-inverted questions like this one, which was fixed for our system.

2. Where do we like to play?
   Chabierski: nothing

   A. Inside
   B. **Outside** ← *last-resort*
   C. At the park

Outside is an adverb, represented as a modifier in the translation, and the system can only answer questions with nouns (nominals) or verb lemmas. This is further explained in Section 7.5.

3. Where do we run and jump?
   Chabierski: nothing

   A. Inside
   B. **In the sun** ← *last-resort*
   C. At the beach

Work is required to represent the question properly in λ-ASP*, as the current translation leads to the erroneous representation

```
answer(W1):-unaryNominal(W0,W1),
     semUnaryEvent(e3,jump,c1),
     semUnaryEvent(e2,run,c1).
```

Story 2

Ron has a pink balloon. He will give it to his mum. She will love it.

1. Who has a balloon?
   Chabierski: ron

   A. Tom

   B. **Ron** ←

The original (*Chabierski*) system was able to handle this question perfectly.

2. What colour is it?
   Chabierski: nothing

   A. Red

   B. **Pink** ← *last-resort*

One problem comes from EasySLR, which orders the arguments for `semBinary(E,be,S,O)` wrongly, switching the subject and object. We also have the issue that 'pink' is an adjective in the text, represented as a modifier, which the system cannot handle as an answer (further explained in Section 7.5).

3. Who will he give it to?
   Chabierski: nothing

   A. **His mum** ← *last-resort*

   B. Ron

The question is translated wrongly, with wrong ordering of arguments and composing two nodes of the CCG parse tree leading to the question predicate `semTernaryEvent(f2,give,W0,f2,c2)` (i.e. `W0` will give event `f2` to him).

Story 3

Dan did not like dogs. Dogs were a danger. Dogs can scare you in the dark. Dan's dad got a dog. Now Dan does like dogs. Dan plays with his dog often. Dan's dog plays in the dirt.

This story did not have any attached questions so the following two are manually crafted.

1. Does Dan like dogs?
   Chabierski: no

   A. **Yes** ←

   B. No

Background knowledge added to define the semantics of 'like' (i.e. one does not stop liking something until otherwise stated) and the semantics of 'not' (changes polarity).

2. Who played in the dirt?
   Chabierski: dog

   A. Dan

   B. **Dan's dog** ←

   C. Dan's dad

The *Chabierski system* was able to answer this question well.

Story 4

Lou plays football. He loves football. Lou can run fast and throw far. He will ask Jon to play with him.

1. What did Lou play?
   Chabierski: football

   A. Games

   B. **Football** ←

   C. Soccer

The *Chabierski system* was able to answer this question well.

2. What can Lou do?
   Chabierski: nothing

   A. **Run fast** ← *last-resort*

   B. Jump high

   C. Score

The question translation for this is

```
answer(W1):-
    unaryNominal(W0,W1),
    binaryModif(can,E0,c1),
    unaryNominal(c1,lou),
    semBinaryEvent(E0,do,X0,W0).
```

This doesn't match with the text translation, which translates 'can do something' to a `unaryModif(can,E)`, not a `binaryModif`. Additionally, the `semBinaryEvent(E0,do,X0,W0)` has the variable `X0` instead of `c1`, so the system does not know whose actions to look for and cannot modify the question body accordingly (see Section 5.2.4 for details on the modifications).

Overall, our system was able to choose the correct answer for every story we tested. Where the choice was made by the *last-resort* algorithm, the system was unable to formulate an answer because of an error caused by an external dependency or because of the handling of `Modifiers`, something that the system needs to improve on.

7.4 Improvements to the *Chabierski System*

The extensions described in Chapter 5 help to elevate the *Chabierski system*, enabling it to answer more questions accurately. The examples given in this evaluation chapter, as well as those in Chapter 5 help to illustrate the enhanced capabilities of our system.

The following are some of the most noteworthy ways we have extended the system:

- **Time Management**
  The improved representation of time points in the text and questions allow for more temporally complex questions to be covered. In this project, we were able to better distinguish between past events (which have happened) and future events (which will happen) and answer questions correctly about them.

- **Coreference Resolution**
  By including more of the coreference annotation provided by CoreNLP, we were able to better gather facts about each entity in a story. This allows for more questions to be answered, as this helps to mitigate errors of argument instantiation in the question translation and helps move away from the requirement of exact matching of question to text.

- **Question Translation**
  We have improved the question translation for specific instances, e.g. future questions, doing questions and those that refer to pronouns not explicitly mentioned in the text, as described in the various subsections of Section 5.2. These help the system answer more questions with higher accuracy.

- **Integration of Commonsense Knowledge**
  We enhance the knowledge added to the ASP program by calling on ConceptNet and including assertions that are deemed relevant to the question and story being considered. This helps the system find answers when the wording of the question does not match that of the text exactly, and also helps to determine the correct option for multiple choice questions.

7.5 Limitations

Aside from inaccuracies that stem from external dependencies that cannot be easily dealt with, there are a number of ways the translation can still be improved. There are outstanding translation tasks, like covering more types of questions, and we can improve the accuracy of answers attained with help from ConceptNet. The following covers some of the areas in which our system is limited and can be developed further.

Accuracy of Free Output Answers

The inclusion of ConceptNet and the rules gathering facts about each entity in a given text makes the system more suited to producing answers for multiple choice questions rather than a free output (i.e. giving the user the answers generated by Clingo). This is because the surplus of information can lead to Clingo finding answers that are connected (but not in the desired way) and thus cluttering the answer with wrong concepts. Having stricter rules in the background knowledge related to ConceptNet relations and in the way we gather facts could help to concentrate the answers generated by Clingo. This is something that we could have developed had we had more time.

Simple Present Tense

The simple present tense in the English language can represent two things: either the action is happening right now (i.e. in that moment) or it happens regularly (or unceasingly), meaning it holds until stated otherwise. This difference is currently indistinguishable by the system, as the CCG category for both is identical. Therefore the system currently assumes all instances of the simple present tense represents an action happening in that moment, and we cannot answer questions which address regular/unceasing occurrences. This is illustrated by Story 4 in Section 7.3, where the first two sentences represent regular actions. We had to modify the questions to past instead of present tense to account for the representation of present tense and rules around it. Since this issue is largely caused by the CCG representation, it is not something that can be easily addressed and will require research into the nuanced differences between the two types of occurrences.

Modifiers: Adjectives and Adverbs

The answers to “where/what is...” can often come from adverbs (e.g. ‘outside’ to describe a location) or adjectives (e.g. ‘pink’ to describe a colour). These are not considered in the current system, which looks for answers to most wh- questions in nominals (with the exception of verbs for doing questions, as explained in Section 5.2.4). We can thus expand the system to look for answers in the modifiers. However, since not all modifiers can be used to answer wh- questions (‘will’, for example, is classified as a modifier) and so we would need a way to verify whether the modifier in question actually matches the type of answer we are looking for. This also relates to the issue discussed below: differentiating wh- questions and verifying the answers given. With more time, we could have looked further into this issue and we believe that an approach to determine which adverbs or adjectives can be used as answers to questions may be to inspect any differences between their CCG categories.

Differentiating Wh- Questions
The system currently does not differentiate between ‘what’ and ‘where’, in that questions like “What am I?” and “Where am I?” both translate to the same ASP representation with `semBinaryEvent(E,be,[I],[answer])`. This can lead to wrong answers generated by Clingo, and an improvement would thus be to do a check after the answer set is found in order to improve the accuracy of answer outputted by the system. This is also an issue that we could have made some progress with had we had more time.

Chapter 8

Conclusion

The aim of this project was to incorporate commonsense knowledge to help enhance machine comprehension of text. This was done by extending the *Chabierski system*, whose question answering portion focused on yes/no questions and who/what/where/which questions with one word answers. We have improved portions of the text translation, question translation and incorporated commonsense knowledge from ConceptNet to further the question answering abilities of the system.

A brief summary of our contributions is as follows:

- Text Translation
  - Improving the representation of time
  - Improving coreference resolution
- Question Translation
  - Generalising arguments of question body predicates to allow for more questions to be answered
  - Improving ASP representation for future tense questions
  - Improving ASP representation of questions about actions (e.g. "What did he do?")
- Enhancing the background knowledge file with rules to complement changes made in the translation
- ConceptNet integration

8.1 Future Work

Machine comprehension of text covers a wide range of tasks, many of which can still be incorporated or improved in our system to enhance both its translation and question-answering abilities. The following sections describe various ways the system can be enhanced.

Relating Phrases

Our integration of ConceptNet currently only allows for connections to be made between *verb* $\leftrightarrow$ *verb*, *noun* $\leftrightarrow$ *noun* or *verb* $\leftrightarrow$ *noun*. This is not always enough, as some concepts need to be expressed as phrases (e.g. *starting a fire* $\leftrightarrow$ *setting kindling ablaze*), but the current ASP representation is not very conducive of making relations between phrases and other representations of concepts.

Translating More Question Types

This is unfortunately limited by the accuracy of the CCG parser, but we can start building λ-ASP expressions for *when* and *why* questions. The CCG representation of some questions can be worked with, and this can be used as a starting point. The use of frames to generate answers in the form of phrases or sentences can also be explored, as *why* questions especially often need to be answered with more than one word.

Allowing Questions from a Different Viewpoint

Many reading comprehension exercises ask questions from a different point of view to the story narrator, especially when the narrator is in first person. This is because the questions are usually asked in third person, a passive voice not representing an entity in the story. It would thus be useful to allow questions that do not match the viewpoint of the story narrator, but this would complicate coreference resolution between the story and the question.

Adding Script Knowledge

Script knowledge, as described in Section 2.3.2, is a type of commonsense knowledge that depicts a typical sequence of actions that form a scenario. This is the type of commonsense knowledge suggested by the group that set SemEval-2018 Task 11, but I found it to be more difficult to incorporate than the more general concepts of ConceptNet and the task of identifying the right scenario for which to add the relevant script knowledge is not a straightforward one. Script knowledge is also useful in very specific cases, whilst ConceptNet has broader applicability. Incorporation of script knowledge would enhance our system's ability to answer more temporal related questions, and so should be added if this is a priority.

Checking Validity of Answers

Since our system currently has the issue of producing 'too many' answers, especially when padded with assertions from ConceptNet, a useful extension to the system would be the implementation of a validity check once answers are produced. For instance, 'where' questions should produce answers which are a location and should not return names of people. This would help to narrow down the answers outputted by Clingo and produce answers of a higher accuracy.

Bibliography

[1] Piotr Chabierski, Alessandra Russo, Mark Law, and Krysia Broda. Machine Comprehension of Text Using Combinatory Categorial Grammar and Answer Set Programs. 2017. pages ii, 2, 3, 15, 23, 25, 59

[2] ConceptNet. `http://www.conceptnet.io/` Date accessed: 2018-01-23. pages ii, 10, 51

[3] A M Turing. COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, 49:433–460, 1950. pages 1

[4] Christopher J C Burges. Towards the Machine Comprehension of Text: An Essay. 2013. pages 1

[5] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. pages 1, 15, 20

[6] Robert Fenner. Alibaba's AI Outguns Humans in Reading Test - Bloomberg, 2018. pages 1

[7] Ernest Davis and Gary Marcus. Commonsense Reasoning and Commonsense Knowledge in Artificial Intelligence. *Communications of the ACM*, sep 2015. pages 1

[8] Yehoshua Bar-Hillel. The Present Status of Automatic Translation of Languages. 1:91–163, 1960. pages 1

[9] Vladimir Lifschitz. What Is Answer Set Programming? pages 5

[10] Mark Law, Alessandra Russo, and Krysia Broda. Inductive Learning of Answer Set Programs. *European Conference on Logics in Artificial Intelligence (JELIA)*, 2(Ray 2009):311–325, 2014. pages 5

[11] Mark Law. Logic-based Learning in ASP. 2017. pages 6

[12] Aravind J. Joshi. Natural Language Processing. pages 6, 7

[13] Noam Chomsky. THREE MODELS FOR THE DESCRIPTION OF LANGUAGE*. 1956. pages 6

[14] Mark Steedman and Jason Baldridge. *Combinatory Categorial Grammar*. Wiley-Blackwell, 2011. pages 7, 8

[15] Mark Steedman. A Very Short Introduction to CCG. 1996. pages 8

[16] John McCarthy. Artificial Intelligence, Logic and Formalizing Common Sense. In *Philosophical Logic and Artificial Intelligence*, pages 161–190. Springer Netherlands, Dordrecht, 1989. pages 9

[17] Hugo Liu and Push Singh. Commonsense Reasoning in and over Natural Language. 2004. pages 10

[18] Robert Speer and Catherine Havasi. Representing General Relational Knowledge in ConceptNet 5. 2012. pages 10, 11

[19] Avron Barr and Edward A. Feigenbaum. Barr, A. and Feigenbaum, E. A. (1981). Frames and scripts. In , pages . Addison-Wesley, California. *The Handbook of Artificial Intelligence*, 3:216–222, 1981. pages 11

[20] Lilian D A Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. DeScript: A Crowdsourced Corpus for the Acquisition of High-Quality Script Knowledge. *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 3494–3501, 2016. pages 12

[21] Michaela Regneri. Learning Script Knowledge with Web Experiments. pages 12, 13

[22] Push Singh, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open Mind Common Sense: Knowledge acquisition from the general public. *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE 2002*, (Davis 1990):1223–1237, 2002. pages 12

[23] Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative event chains. *Proceedings of the Association of Computational Linguistics*, 31(14):789–797, 2008. pages 13, 14, 19

[24] Nathanael Chambers and Dan Jurafsky. Unsupervised Learning of Narrative Schemas and their Participants. *Acl*, (August):602–610, 2009. pages 13

[25] Ashutosh Modi and Ivan Titov. Inducing Neural Models of Script Knowledge. *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 49–57, 2014. pages 13, 14

[26] Lilian D A Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. *Inducing Script Structure from Crowdsourced Event Descriptions via Semi-Supervised Clustering*. 2017. pages 14

[27] SemEval-2018 Task 11: Machine Comprehension using Commonsense Knowledge. `https://competitions.codalab.org/competitions/17184` Date accessed: 2018-01-23. pages 14, 23, 32, 49

[28] Chitta Baral, Juraj Dzifcak, and Tran Cao Son. Using Answer Set Programming and Lambda Calculus to Characterize Natural Language Sentences with Normatives and Exceptions. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 2:818–823, 2008. pages 15

[29] Irene-Anna Diakidoy, Antonis Kakas, Loizos Michael, and Rob Miller. STAR: A System of Argumentation for Story Comprehension and Beyond. 2015. pages 17, 18

[30] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A Corpus and Cloze Evaluation for Deeper Understanding of Commonsense Stories. 2016. pages 19

[31] Hongyu Lin, Le Sun, and Xianpei Han. Reasoning with Heterogeneous Knowledge for Commonsense Machine Comprehension. pages 2032–2043, 2017. pages 19

[32] *Learning Deep Structured Semantic Models for Web Search using Clickthrough Data*. ACM International Conference on Information and Knowledge Management (CIKM), oct 2013. pages 19

[33] Karl Pichotta and Raymond J Mooney. Learning Statistical Scripts with LSTM Recurrent Neural Networks. *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016. pages 20

[34] Matthew Richardson, Christopher J C Burges, and Erin Renshaw. MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text. pages 193–203, 2013. pages 20

[35] Karl Moritz Hermann, Tomáš Kočisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching Machines to Read and Comprehend. 2015. pages 20

[36] Wilson L. Taylor. cloze procedure: A new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433, 1953. pages 20

[37] SQuAD - the Stanford Question Answering Dataset. `https://rajpurkar.github.io/SQuAD-explorer/` Date accessed: 2018-01-25. pages 20, 21

[38] Robin Jia and Percy Liang. Adversarial Examples for Evaluating Reading Comprehension Systems. pages 21

[39] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional Attention Flow for Machine Comprehension. *CoRR*, abs/1611.01603, 2016. pages 21

[40] Shuohang Wang and Jing Jiang. Machine Comprehension Using Match-LSTM and Answer Pointer. *CoRR*, abs/1608.07905, 2016. pages 21

[41] W. N. Francis and H. Kucera. Brown Corpus Manual. Technical report, Brown University, 1979. pages 21

[42] Liang Wang, Meng Sun, Wei Zhao, Kewei Shen, and Jingming Liu. Yuanfudao at SemEval-2018 Task 11: Three-way Attention and Relational Knowledge for Commonsense Machine Comprehension. pages 22

[43] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. TOWARDS AI-COMPLETE QUESTION ANSWERING: A SET OF PREREQUISITE TOY TASKS. 2015. pages 23

[44] Piotr Chabierski. Logic-based Approach to Machine Comprehension of Text. 2017. pages 24, 42

[45] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David Mcclosky. The Stanford CoreNLP Natural Language Processing Toolkit. pages 25, 26

[46] Mike Lewis, Luheng He, and Luke Zettlemoyer. Joint A * CCG Parsing and Semantic Role Labeling. pages 25, 32

[47] Penn Treebank P.O.S. Tags. pages 25

[48] Mike Lewis, Kenton Lee, and Luke Zettlemoyer. LSTM CCG Parsing. pages 221–231. pages 25, 26

[49] clingo - A grounder and solver for logic programs. `https://github.com/potassco/clingo` Date accessed: 2018-06-05. pages 25

[50] ConceptNet 5 FAQ — GitHub. pages 51

[51] A Wellspring of Worksheets Exercise. `https://homeshealth.info/kindergarten-reading-comprehension-worksheets.html/pleasant-kindergarten-reading-comprehension-worksheets-also-kindergarten-reading-preh`
Date accessed: 2018-06-14. pages 93

[52] CRIA Books Reading Comprehension Exercise. `https://homeshealth.info/kindergarten-reading-comprehension-worksheets.html/endearing-kindergarten-reading-comprehension-worksheets-in-kindergarten-reading-prehe`
Date accessed: 2018-06-14. pages 94

[53] Have Fun Teaching Alphabet Reading Comprehension Stories: D. `https://homeshealth.info/kindergarten-reading-comprehension-worksheets.html/useful-kindergarten-reading-comprehension-worksheets-for-kindergarten-reading-prehens`
Date accessed: 2018-06-14. pages 95

[54] CRIA Books Reading Comprehension Exercise. `http://criabooks.com/reading-comprehension-for-kindergarten-worksheets/kindergarten-reading-comprehension-passages-set-1-worksheets-pdf-2481cd49231caffadbcc`
Date accessed: 2018-06-14. pages 96

Appendix A

Original Background Knowledge

File A.1 Background Knowledge Used by Chabierski et al. for bAbI Dataset

```
% Unary.
semUnaryEvent(E,L,Y) :- unaryFluent(E,L,Y).
unaryFluent(E,L,Y) :- unaryInitEvent(E,L,Y).
unaryFluent(E,L,Y) :- unaryFluent(E1,L,Y), previous(E1,E),
    not unaryTermEvent(E,L,Y).

% Binary.
semBinaryEvent(E,L,Y,Z) :- binaryFluent(E,L,Y,Z).
binaryFluent(E,L,Y,Z) :- binaryInitEvent(E,L,Y,Z).
binaryFluent(E,L,Y,Z) :- binaryFluent(E1,L,Y,Z), previous(E1,E),
    not binaryTermEvent(E,L,Y,Z).

% Time points defined by the mata predicates.
previous(E1,E) :- metaData(T1,E1), metaData(T,E), T=T1+1.

% Appendix - mapping to 'semantic' predicates.
semTernaryEvent(E,L,X,Y,Z) :- ternaryEvent(E,L,X,Y,Z),
    not abTernaryEvent(E,L,X,Y,Z).
semBinaryEvent(E,L,Y,Z) :- binaryEvent(E,L,Y,Z),
    not abBinaryEvent(E,L,Y,Z).
semUnaryEvent(E,L,Z) :- unaryEvent(E,L,Z), not abUnaryEvent(E,L,Z).

unaryNominal(X,Y) :- eq(X,Z), unaryNominal(Z,Y).
eq(X,Y) :- eq(Y,X).


% Rules about the change of position of an entity.
binaryInitEvent(E,be,P,L) :- binaryEvent(E,go,P,L).
binaryTermEvent(E,be,P,L) :- binaryEvent(E,go,P,L2),
    unaryNominal(L,C).

% Event Calculus rules stating when object is carried by an entity.
binaryInitEvent(V0,carry,V1,V2) :- semBinaryEvent(V0,take,V1,V2).
```

```
binaryTermEvent(V0,carry,V1,V2) :- semBinaryEvent(V0,take_out,V1,V2).

% Rules saying that an object changes its position together
% with a person that carries it.
binaryEvent(E,be,O,L) :- semBinaryEvent(E,carry,P,O),
    semBinaryEvent(E,be,P,L).
binaryInitEvent(E,be,O,L) :- binaryTermEvent(E,carry,P,O),
    semBinaryEvent(E,be,P,L).
binaryTermEvent(E,be,O,L) :- semBinaryEvent(E,carry,P,O),
    binaryTermEvent(E,be,P,L).

% Rules regarding the change of possession of objects.
binaryEvent(E,receive,R,O) :- ternaryEvent(E,bring,G,O,R).
binaryEvent(E,bring,G,O) :- ternaryEvent(E,bring,G,O,R).

% Semantics of forget - forget changes polarity.
abBinaryEvent(E,L,X,Y) :- binaryEvent(E,L,X,Y),
    binaryEvent(E1,forget,X,E).

% Temporal rules, semantics of 'before'.
binaryPrep(before,E0,E1) :- previous(E0,E1).
binaryPrep(before,E0,E2) :- binaryPrep(before,E1,E2),
    previous(E0,E1).
```

Appendix B

Background Knowledge

File B.1 General Background Knowledge Rules

```
% Persistance rules motivated by event calculus.

% Unary.
semUnaryEvent(E,L,Y) :- unaryFluent(E,L,Y).
unaryFluent(E,L,Y) :- unaryInitEvent(E,L,Y).
unaryFluent(E,L,Y) :- unaryFluent(E1,L,Y), previous(E1,E),
    not unaryTermEvent(E,L,Y).

% Binary.
semBinaryEvent(E,L,Y,Z) :- binaryFluent(E,L,Y,Z).
binaryFluent(E,L,Y,Z) :- binaryInitEvent(E,L,Y,Z).
binaryFluent(E,L,Y,Z) :- binaryFluent(E1,L,Y,Z), previous(E1,E),
    not binaryTermEvent(E,L,Y,Z).

% Time points defined by the meta predicates.
previous(E1,E2) :- metaData(T1,E1,X), metaData(T2,E2,X), T2=T1+1.
previous(E1,E2) :- metaData(T1,E1,past), metaData(0,E2,present),
    last(E1).
previous(E1,E2) :- metaData(T1,E1,present), metaData(0,E2,future),
    last(E1).

last(E) :- metaData(T,E,X), not metaData(T+1,_,X).

% Appendix - mapping to 'semantic' predicates.
semTernaryEvent(E,L,X,Y,Z) :- ternaryEvent(E,L,X,Y,Z),
    not abTernaryEvent(E,L,X,Y,Z).
semBinaryEvent(E,L,Y,Z) :- binaryEvent(E,L,Y,Z),
    not abBinaryEvent(E,L,Y,Z).
semUnaryEvent(E,L,Z) :- unaryEvent(E,L,Z),
    not abUnaryEvent(E,L,Z).
semZeroEvent(E,L) :- zeroEvent(E,L), not abZeroEvent(E,L).

% Semantics of not
```

```
abUnaryEvent(E,L,Z) :- unaryEvent(E,L,Z), unaryModif(escnot,E).
abBinaryEvent(E,L,X,Y) :- binaryEvent(E,L,X,Y),
    unaryModif(escnot,E).
abTernaryEvent(E,L,X,Y,Z) :- ternaryEvent(E,L,X,Y,Z),
    unaryModif(escnot,E).

unaryNominal(X,Y) :- eq(X,Z), unaryNominal(Z,Y).
eq(X,Y) :- eq(Y,X).

% Rules about the change of position of an entity.
binaryInitEvent(E,be,P,L) :- binaryEvent(E,go,P,L).
binaryTermEvent(E,be,P,L) :- binaryEvent(E,go,P,L2),
    unaryNominal(L,C).

% Semantics of be and like (i.e. you don't stop being and liking
% something from one time point to another unless something
% affects it

binaryInitEvent(E,be,P,L) :- binaryEvent(E,be,P,L),
    not abBinaryEvent(E,be,P,L).
binaryTermEvent(E,be,P,L) :- semBinaryEvent(E,be,P,L),
    unaryModif(escnot,E).

binaryInitEvent(E,like,S,O) :- binaryEvent(E,like,S,O),
    not abBinaryEvent(E,like,S,O).
binaryTermEvent(E,like,S,O) :- semBinaryEvent(E,like,S,O),
    unaryModif(escnot,E).

% If an object is inside a location, it is in that location
% (the format of this is quite specific to the language of a
% specific story and could perhaps be more generalised)
binaryInitEvent(E,be,P,L) :- binaryPrep(inside,P,L),
    binaryPrep(I,in,E,P).

% If an object is placed in a location, is is in that location
binaryInitEvent(E,be,P,L) :- binaryPrep(I,in,E,L),
    binaryEvent(E,place,C,P).

% Transitivity of being in a location
binaryInitEvent(E1,be,P,L2) :- semBinaryEvent(E1,be,P,L1),
    semBinaryEvent(E2,be,L1,L2).

% Event Calculus rules stating when object is carried by an entity.
binaryInitEvent(V0,carry,V1,V2) :- semBinaryEvent(V0,take,V1,V2).
binaryTermEvent(V0,carry,V1,V2) :- semBinaryEvent(V0,take_out,V1,V2).

% Rules saying that an object changes its position together with a person that
% carries it.
binaryEvent(E,be,O,L) :- semBinaryEvent(E,carry,P,O),
    semBinaryEvent(E,be,P,L).
binaryInitEvent(E,be,O,L) :- binaryTermEvent(E,carry,P,O),
    semBinaryEvent(E,be,P,L).
binaryTermEvent(E,be,O,L) :- semBinaryEvent(E,carry,P,O),
    binaryTermEvent(E,be,P,L).

% Rules regarding the change of posession of objects.
binaryEvent(E,receive,R,O) :- ternaryEvent(E,bring,G,O,R).
binaryEvent(E,bring,G,O) :- ternaryEvent(E,bring,G,O,R).

% Semantics of forget - forget changes polarity.
abBinaryEvent(E,L,X,Y) :- binaryEvent(E,L,X,Y), binaryEvent(E1,forget,X,E).

% Temporal rules, semantics of 'before'.
binaryPrep(before,E0,E1) :- previous(E0,E1).
binaryPrep(before,E0,E2) :- binaryPrep(before,E1,E2), previous(E0,E1).

% Semantics of arriving and leaving
binaryInitEvent(E,be,P,L) :- semBinaryEvent(E,arrive,P,L).
binaryTermEvent(E,be,P,L) :- semBinaryEvent(E,leave,P,L).

% Generalised notion that someone's name is an identifier for them
% e.g 'my name is X' means 'X is an identifier for I'
unaryNominal(C2, X) :- binaryEvent(E0,be,N0,C1),
    unaryNominal(C1,X), binaryModif(poss,C2,N0),
    unaryNominal(N0,name), unaryNominal(C2, _).

% Semantics of have
binaryInitEvent(E,have,S,O) :- semBinaryEvent(E,have,S,O).
binaryTermEvent(E,have,S,O) :- semBinaryEvent(E,lose,S,O).

% Possession means I have
binaryEvent(E,have,S,O) :- binaryModif(poss,S,O),
    binaryEvent(E,_,O,_).
binaryEvent(E,have,S,O) :- binaryModif(poss,S,O),
    binaryEvent(E,_,_,O).
binaryModif(poss,S,O) :- binaryEvent(_,have,S,O).

% Notion of "If A uses X to do Y, A uses X"
semBinaryEvent(E0,use,C1,E1) :- semTernaryEvent(E0,use,C1,E1,_).

% Notion of "If A uses X to do Y, A does Y"
semBinaryEvent(E1,V1,C1,N1) :- semTernaryEvent(E0,use,C1,N0,E1),
    semBinaryEvent(E1,V1,N0,N1).

% Notion of "If A [verb] B, A [verb]"
semUnaryEvent(E0,V1,C1) :- semBinaryEvent(E0,V1,C1,_).

% Collecting information for one noun
```

```
semUnaryEvent(E,V,X0) :- semUnaryEvent(E,V,X1),
    unaryNominal(X1,W), unaryNominal(X0,W).

semBinaryEvent(E,V,X0,Y) :- semBinaryEvent(E,V,X1,Y),
    unaryNominal(X1,W), unaryNominal(X0,W).
semBinaryEvent(E,V,X,Y0) :- semBinaryEvent(E,V,X,Y1),
    unaryNominal(Y1,W), unaryNominal(Y0,W).

semTernaryEvent(E,V,X0,Y,Z) :- semTernaryEvent(E,V,X1,Y,Z),
    unaryNominal(X1,W), unaryNominal(X0,W).
semTernaryEvent(E,V,X,Y0,Z) :- semTernaryEvent(E,V,X,Y1,Z),
    unaryNominal(Y1,W), unaryNominal(Y0,W).
semTernaryEvent(E,V,X,Y,Z0) :- semTernaryEvent(E,V,X,Y,Z0),
    unaryNominal(Z1,W), unaryNominal(Z0,W).

% This is specifically because SemEval questions have answers that are 'narrato
answer(narrator) :- answer(i).
answer(narrator) :- answer(we).

% Semantic rule for future events
abBinaryEvent(E,L,X,Y) :- metaData(_,E,future),
    binaryEvent(E,L,X,Y).

% If question and relevant text info are both future, should be satisfied
semBinaryEvent(E1,V,X,Y) :- metaData(T1,E1,future),
    metaData(T0,E0,future), binaryEvent(E0,V,X,Y),
    binaryPrep(before,E0,E1).

% ConceptNet
related(X,Y) :- conceptNetRelation(_,X,Y).
related(X,Y) :- conceptNetRelation(_,Y,X).
related(X,Z) :- related(X,Y), related(Y,Z).

semUnaryEvent(E,V,S) :- related(V,V1), semUnaryEvent(E,V1,S).
semBinaryEvent(E,V,S,O) :- related(V,V1),
    semBinaryEvent(E,V1,S,O).
semTernaryEvent(E,V,S,O,W) :- related(V,V1),
    semTernaryEvent(E,V1,S,O,W).
unaryNominal(I,W0) :- related(W0,W1), unaryNominal(I,W1).
```

Appendix C

ConceptNet Relations

Notes:

- A and B refer to the start and end nodes (of form A → B) for each relation.
- All relations can be prefixed with 'Not' to express a negative assertion.

#	Relation	Description	Used?
1	RelatedTo	Some unknown positive relation between A and B	Not used in our system as it is too vague
2	ExternalURL	A URL outside of ConceptNet where further Linked Data about the concept can be found	Not used in our system
3	FormOf	B is the root word of A (e.g. *sleep* $\xrightarrow{\text{FormOf}}$ *slept*)	Stored & used in our system
4	IsA	A is a subtype or instance of B (e.g. *apple* $\xrightarrow{\text{IsA}}$ *fruit* (e.g. *Chicago* $\xrightarrow{\text{IsA}}$ *city*))	Stored & used in our system
5	PartOf	A is a part of B (e.g. *gearshift* $\xrightarrow{\text{PartOf}}$ *car*)	Stored & used in our system
6	HasA	B belongs to A often the reverse of PartOf (e.g. *bird* $\xrightarrow{\text{HasA}}$ *wing*)	Stored & used in our system
7	UsedFor	A is used for B (e.g. *bridge* $\xrightarrow{\text{UsedFor}}$ *cross water*)	Stored & used in our system
8	CapableOf	A can typically do B (e.g. *knife* $\xrightarrow{\text{CapableOf}}$ *cut*)	Stored & used in our system
9	AtLocation	B is a typical/inherent location for A (e.g. *butter* $\xrightarrow{\text{AtLoc}}$ *fridge* *Boston* $\xrightarrow{\text{AtLoc}}$ *Massachusetts*)	Stored & used in our system

#	**Relation**	**Description**	**Used?**
10	Causes	A and B are events A typically causes B (e.g. *exercise* $\xrightarrow{\text{Causes}}$ *sweat*)	Stored & used in our system
11	HasSubevent	A and B are events B happens as subevent of A (e.g. *eating* $\xrightarrow{\text{HasSubevent}}$ *chewing*)	Stored but not really used in our system
12	HasFirstSubevent	A is an event that starts with subevent B (e.g. *sleep* $\xrightarrow{\text{HFS}}$ *close eyes*)	Stored but not really used in our system
13	HasLastSubevent	A is an event that ends with subevent B (e.g. *cook* $\xrightarrow{\text{HLS}}$ *clean kitchen*)	Stored but not really used in our system
14	HasPrerequisite	B needs to happen in order for A to happen (e.g. *dream* $\xrightarrow{\text{HasPre}}$ *sleep*)	Stored & used in our system
15	HasProperty	A can be described by B (e.g. *ice* $\xrightarrow{\text{HasProp}}$ *cold*)	Stored & used in our system
16	MotivatedByGoal	A is a step towards accomplishing B (e.g. *compete* $\xrightarrow{\text{MBG}}$ *win*)	Stored & used in our system
17	ObstructedBy	A is prevented by B (e.g. *sleep* $\xrightarrow{\text{ObstBy}}$ *noise*)	Stored but not used in our system
18	Desires	A is a **conscious** entity that typically wants B (e.g. *person* $\xrightarrow{\text{Desires}}$ *love*)	Stored & used in our system
19	CreatedBy	B is a process or agent that creates A (e.g. *cake* $\xrightarrow{\text{CreatedBy}}$ *bake*)	Stored & used in our system
20	Synonym	A and B are synonyms (e.g. *sunlight* $\xrightarrow{\text{Synonym}}$ *sunshine*)	Stored & used in our system
21	Antonym	A and B are opposites (e.g. *hot* $\xrightarrow{\text{Antonym}}$ *cold*)	Stored but not used in our system
22	DistinctFrom	A and B are distinct members of a set (e.g. *August* $\xrightarrow{\text{DistinctFrom}}$ *November*)	Stored but not used in our system
23	DerivedFrom	B appears in A and contributes to A's meaning (e.g. *pocketbook* $\xrightarrow{\text{DerFrom}}$ *book*)	Stored & used in our system
24	SymbolOf	A symbolically represents B (e.g. *red* $\xrightarrow{\text{SymbolOf}}$ *fervor*)	Stored & used in our system

#	**Relation**	**Description**	**Used?**
25	DefinedAs	B is a more exploratory version of A (e.g. *peace* $\xrightarrow{\text{DefAs}}$ *absence of war*)	Stored & used in our system
26	Entails	If A happens, B happens (e.g. *run* $\xrightarrow{\text{Entails}}$ *move*)	Stored & used in our system
27	MannerOf	A is a way to do B (for verbs) (e.g. *auction* $\xrightarrow{\text{MannerOf}}$ *sale*)	Stored & used in our system
28	LocatedNear	A and B typically found near each other (e.g. *chair* $\xrightarrow{\text{LocNear}}$ *table*)	Stored but not really used in our system
29	HasContext	A is used in context of B (e.g. *arvo* $\xrightarrow{\text{HasContext}}$ *Australia*)	Stored but not used in our system (too vague)
30	dbpedia	Imported from dbpedia doesn't correspond to any existing relations	Not used in our system
31	SimilarTo	A is similar to B (e.g. *mixed* $\xrightarrow{\text{ST}}$ *food processor*)	Stored & used in our system
32	Etymologically RelatedTo	A and B have a common origin (e.g. *folkmusiikki* $\xrightarrow{\text{ERT}}$ *folk music*)	Stored but not used in our system
33	Etymologically DerivedFrom	A is derived from B (e.g. *detja* $\xrightarrow{\text{EDF}}$ *date*)	Stored but not used in our system
34	CausesDesire	A makes someone want B (e.g. *have no food* $\xrightarrow{\text{CD}}$ *go to store*)	Stored & used in our system
35	MadeOf	A is made of B (e.g. *bottle* $\xrightarrow{\text{MadeOf}}$ *plastic*)	Stored & used in our system
36	ReceivesAction	B can be done to A (e.g. *button* $\xrightarrow{\text{RA}}$ *push*)	Stored & used in our system
37	InstanceOf	A is example of B (e.g. *meringue* $\xrightarrow{\text{InstanceOf}}$ *dessert*)	Stored & used in our system

Appendix D

SemEval-2018 Task 11 Stories

File D.1 Narratives from the Development Set Provided by SemEval-2018 Task 11

```
<data>
  <instance id="13" scenario="ordering a pizza">
    <text>
      One afternoon when I was visiting with friends, we suddenly
      got very hungry!  So, we decided to order a pizza.  I
      looked in the telephone book and found several pizza
      restaurants that would deliver.  We all talked it over and
      settled on a pizza from Domino's.  Then we discussed what
      we would like on our pizza and what size we should get.  I
      called the number and gave our order to the person who
      answered.  We ordered a large pepperoni and onion pizza
      with extra cheese.  He told me that our pizza would cost
      $15.00.  I said that would be fine and asked if the pizza
      could be delivered.  He said, &quot;yes&quot; and asked for
      my address.  I then gave him my name, address, and phone
      number.  He told me that our pizza would be delivered in 30
      minutes.  When the delivery boy arrived, we paid for our
      pizza and included a $2.00 tip.  Then we enjoyed eating our
      pizza!
    </text>
    <questions>
      <question id="0" text="When will it be here?" type="text">
        <answer correct="False" id="0" text="It took an hour."/>
        <answer correct="True" id="1" text="It took 30 minutes."/>
      </question>
      <question id="1" text="Who ordered the pizza?" type="text">
        <answer correct="True" id="0" text="The narrator"/>
        <answer correct="False" id="1" text="The dog."/>
      </question>
      <question id="2" text="Did they order any other entrees?"
       type="text">
        <answer correct="True" id="0" text="No"/>
        <answer correct="False" id="1" text="Yes"/>
      </question>
    <question id="3" text="Where did they order it?" type="text">
      <answer correct="False" id="0" text="Pizza Hut"/>
      <answer correct="True" id="1" text="from Domino's"/>
    </question>
    <question id="4" text="Would they order the same thing
     again?" type="commonsense">
      <answer correct="False" id="0" text="no"/>
      <answer correct="True" id="1" text="yes, it was good"/>
    </question>
    <question id="5" text="Did they use a coupon?"
     type="commonsense">
      <answer correct="True" id="0" text="No"/>
      <answer correct="False" id="1" text="Yes"/>
    </question>
    <question id="6" text="How long did they have to wait for the
     meal?" type="text">
      <answer correct="True" id="0" text="30 minutes"/>
      <answer correct="False" id="1" text="15 minutes"/>
    </question>
    <question id="7" text="Which restaurant did they order the
     pizza from?" type="text">
      <answer correct="True" id="0" text="Domino's"/>
      <answer correct="False" id="1" text="Pizza Hut"/>
    </question>
    <question id="8" text="What door did the pizza get delivered
     to?" type="commonsense">
      <answer correct="False" id="0" text="it didn't say."/>
      <answer correct="True" id="1" text="The front door"/>
    </question>
    <question id="9" text="How much of a tip was given?"
     type="text">
      <answer correct="False" id="0" text="$5.00"/>
      <answer correct="True" id="1" text="$2.00"/>
    </question>
  </questions>
</instance>

<instance id="15" scenario="making a bonfire">
  <text>
    My family and I decided that the evening was beautiful so
    we wanted to have a bonfire.  First, my husband went to our
    shed and gathered some dry wood.  I placed camp chairs around
    our fire pit.  Then my husband placed the dry wood in a
    pyramid shape inside the fire pit.  He then set some small
    kindling ablaze using a long lighter.  Once the kindling
    reached the dry wood, it set the wood on fire.  We then sat
    around the fire for some time, adding more logs as the
    previous ones burned out.  We cooked marshmallows on sticks
    over the open flames.  When the marshmallows were golden
```

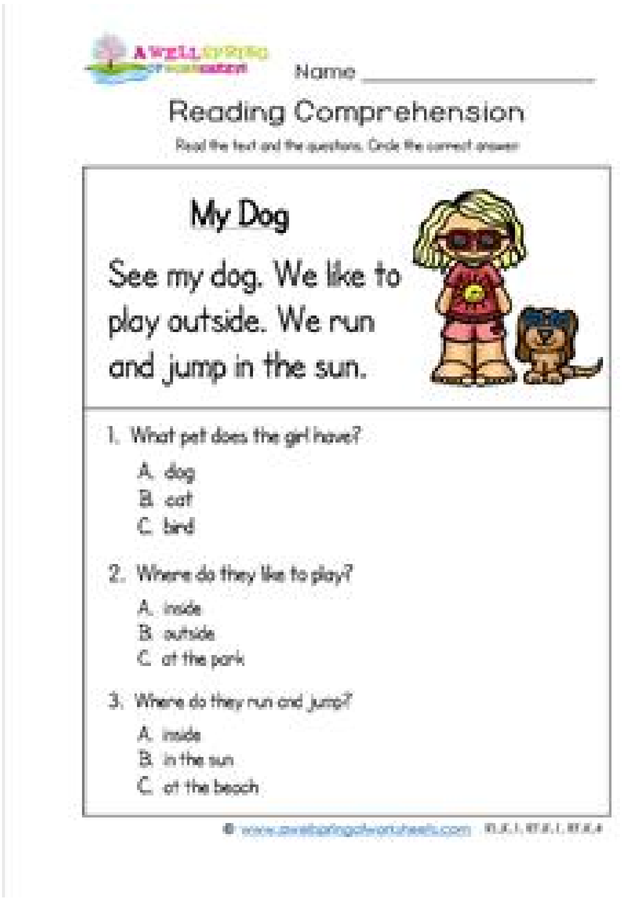
```
      brown, we placed them between two graham crackers with
      chocolate pieces.  We ate our S'mores as we joked, laughed
      and told stories around our beautiful fire.  When we
      finished, I put away our camp chairs.  My husband made sure
      the fire was out by dousing it with some water and we went
      inside to bed.
    </text>
    <questions>
      <question id="0" text="Who started the fire?" type="text">
        <answer correct="False" id="0" text="Wife."/>
        <answer correct="True" id="1" text="Their husband lit
          it."/>
      </question>
      <question id="1" text="Where did they make the bonfire?"
       type="text">
        <answer correct="False" id="0" text="in the garage"/>
        <answer correct="True" id="1" text="Their fire pit"/>
      </question>
      <question id="2" text="How long did it take to build the
       fire?" type="commonsense">
        <answer correct="True" id="0" text="Few minutes."/>
        <answer correct="False" id="1" text="Hours"/>
      </question>
      <question id="3" text="Who built the bonfire?" type="text">
        <answer correct="False" id="0" text="the wife"/>
        <answer correct="True" id="1" text="Their husband."/>
      </question>
      <question id="4" text="Had they ever built a bonfire before
       this one?" type="commonsense">
        <answer correct="True" id="0" text="Yes, they had built one
          before?"/>
        <answer correct="False" id="1" text="No, this was their
          first."/>
      </question>
      <question id="5" text="When did they put the logs in the fire
       pit?" type="text">
        <answer correct="False" id="0" text="They did it the next
          morning."/>
        <answer correct="True" id="1" text="After they gathered all
          the logs."/>
      </question>
      <question id="6" text="Where did they build the bonfire?"
       type="text">
        <answer correct="True" id="0" text="Their fire pit"/>
        <answer correct="False" id="1" text="in the house"/>
      </question>
      <question id="7" text="What did they do at the bonfire?"
       type="text">
        <answer correct="True" id="0" text="Talk and cook
```

```
          smores."/>
        <answer correct="False" id="1" text="They were dancing
          around."/>
      </question>
    </questions>
  </instance>

  <instance id="18" scenario="cleaning up a flat">
    <text>
      I needed to clean up my flat. I had to get my broom and
      vacuum and all the cleaners I would need. I started by going
      around and picking up any garbage I seem, like used candy
      wrappers and old bottles of water.I threw them away. I went
      around and picked up and dishes that were out and put them in
      the sink then washed them. I used my broom to sweep up all
      the dust and dirt off the floors the hard floors, and the
      vacuum to clean up the floors that had rugs and carpets. When
      the hard floors were swept, I used a bucket with floor
      cleaner and water, and a mop, to mop them up. As the dried I
      took a rag and began dusting everything in the room. The
      t.v., the tables and counters and everything else that was a
      hard surface, I used the rag to dust. My flat looked very
      nice when it was clean.
    </text>
    <questions>
      <question id="0" text="Did they use a mop?" type="text">
        <answer correct="False" id="0" text="No"/>
        <answer correct="True" id="1" text="Yes after sweeping"/>
      </question>
      <question id="1" text="Is it hard work to clean the flat?"
       type="commonsense">
        <answer correct="True" id="0" text="No."/>
        <answer correct="False" id="1" text="yes"/>
      </question>
      <question id="2" text="Where did they vacuum?" type="text">
        <answer correct="False" id="0" text="Hardwood floor"/>
        <answer correct="True" id="1" text="Rooms with rugs and
          carpet"/>
      </question>
      <question id="3" text="Why did the flat needed cleaning?"
       type="commonsense">
        <answer correct="True" id="0" text="It was a big mess."/>
        <answer correct="False" id="1" text="The dishes were washed
          at the sink."/>
      </question>
    </questions>
  </instance>
</data>
```

Appendix E

Kindergarten Reading Comprehension



Name ____________________

Reading Comprehension

Read the text and the questions. Circle the correct answer.

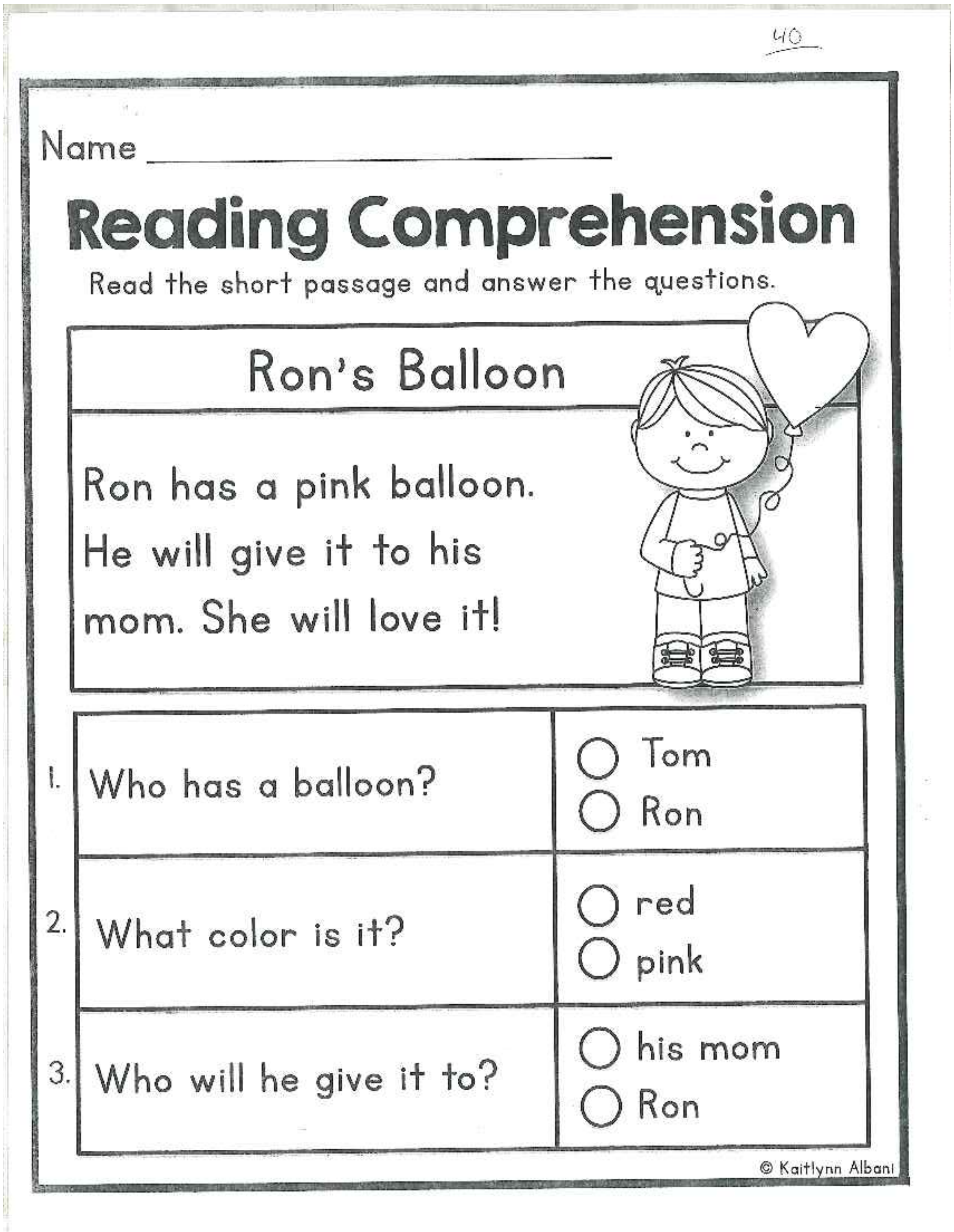
My Dog

See my dog. We like to play outside. We run and jump in the sun.

1. What pet does the girl have?
   - A. dog
   - B. cat
   - C. bird
2. Where do they like to play?
   - A. inside
   - B. outside
   - C. at the park
3. Where do they run and jump?
   - A. inside
   - B. in the sun
   - C. at the beach

© www.aWellspringofworksheets.com RL.K.1, RF.K.1, RF.K.4

Figure E.1: Exercise from A Wellspring of Worksheets [51]



40

Name ____________________

Reading Comprehension

Read the short passage and answer the questions.

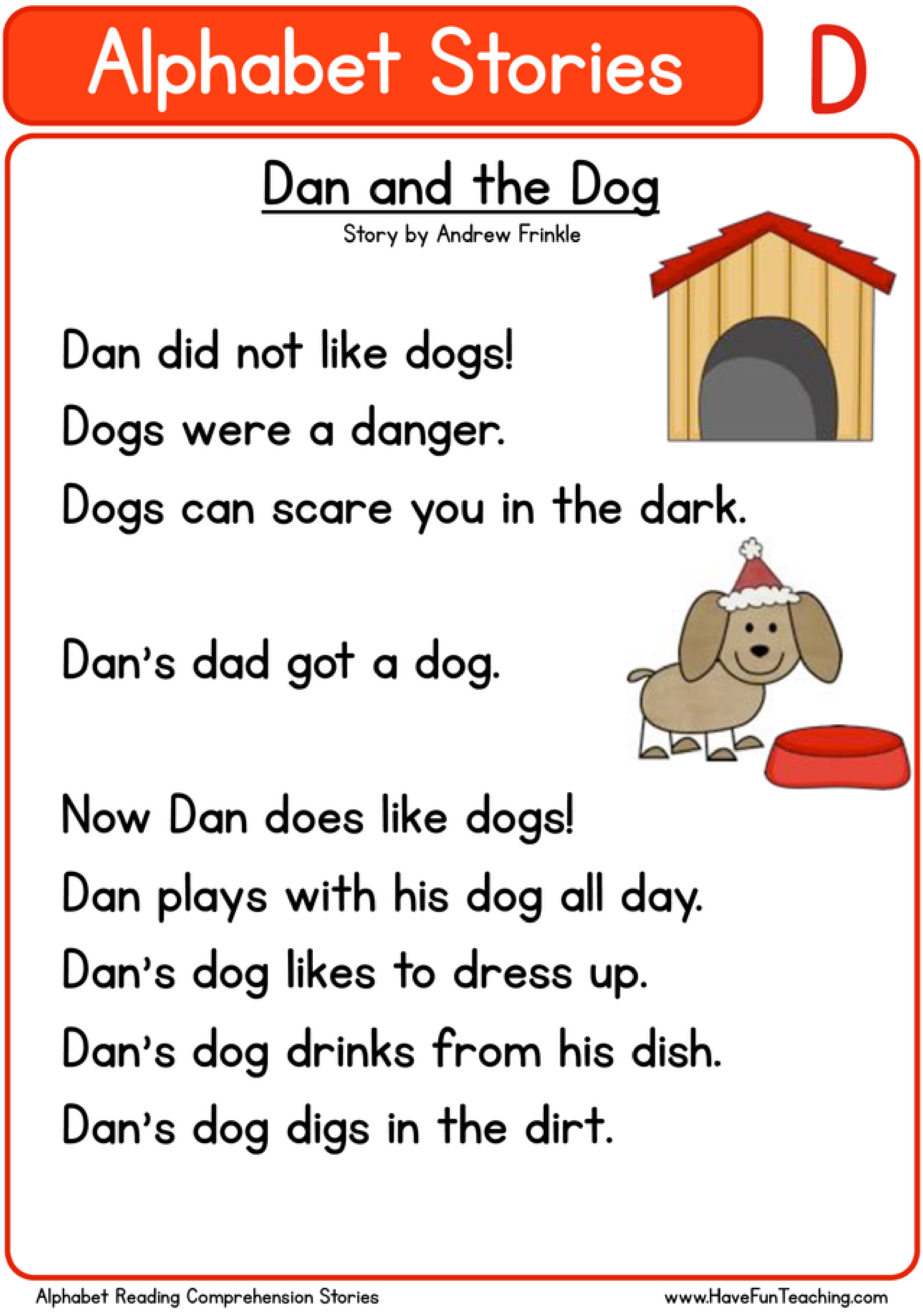
Ron's Balloon

Ron has a pink balloon. He will give it to his mom. She will love it!

1.	Who has a balloon?	○ Tom ○ Ron
2.	What color is it?	○ red ○ pink
3.	Who will he give it to?	○ his mom ○ Ron

© Kaitlynn Albani

Figure E.2: Exercise from homeshealth.info [52]



Alphabet Stories D

Dan and the Dog

Story by Andrew Frinkle

Dan did not like dogs!
Dogs were a danger.
Dogs can scare you in the dark.

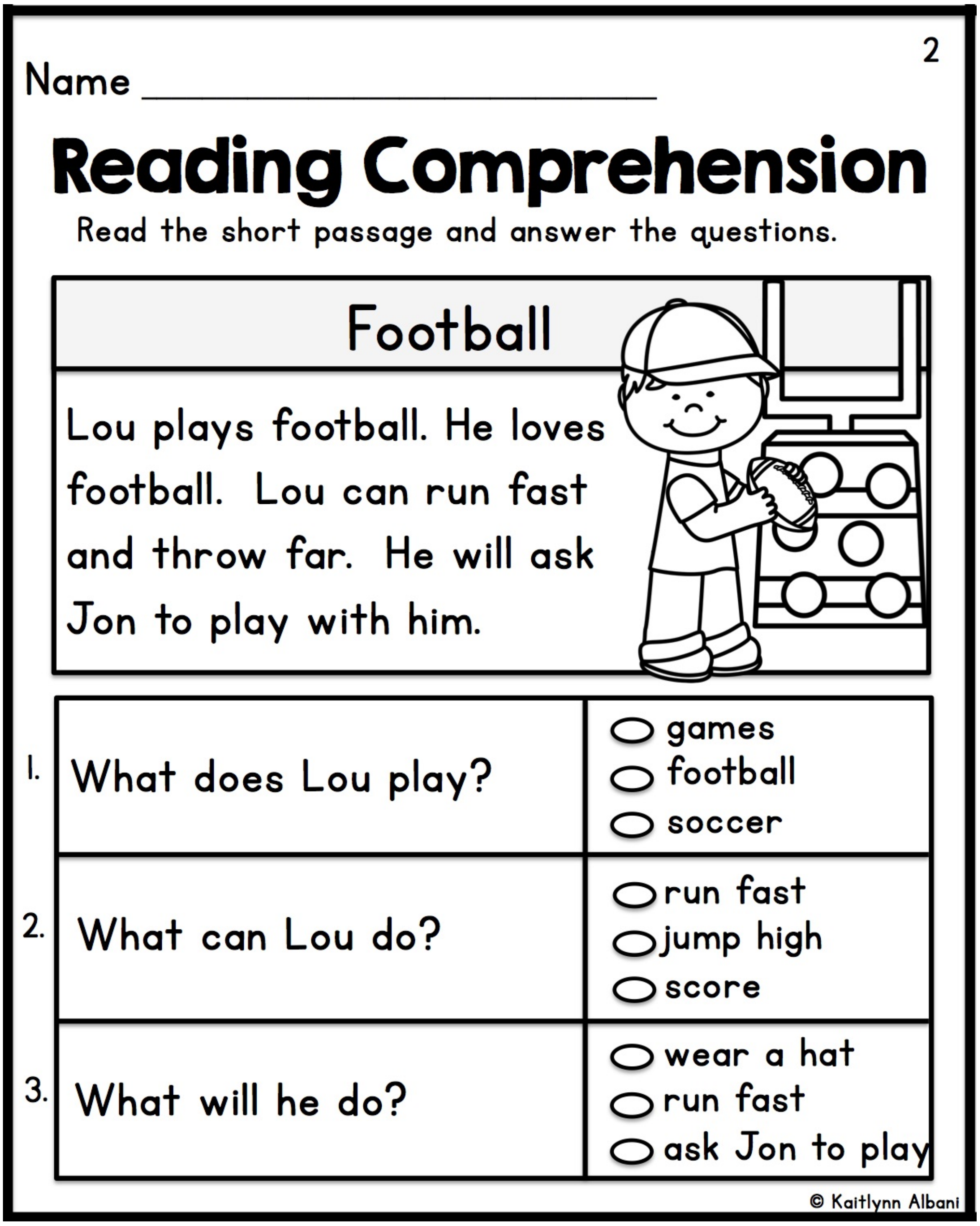
Dan's dad got a dog.

Now Dan does like dogs!
Dan plays with his dog all day.
Dan's dog likes to dress up.
Dan's dog drinks from his dish.
Dan's dog digs in the dirt.

Alphabet Reading Comprehension Stories

www.HaveFunTeaching.com

Figure E.3: Exercise from Have Fun Teaching [53]



2

Name ______________________

Reading Comprehension

Read the short passage and answer the questions.

Football

Lou plays football. He loves football. Lou can run fast and throw far. He will ask Jon to play with him.

1.	What does Lou play?	○ games ○ football ○ soccer
2.	What can Lou do?	○ run fast ○ jump high ○ score
3.	What will he do?	○ wear a hat ○ run fast ○ ask Jon to play

© Kaitlynn Albani

Figure E.4: Exercise from CRIA Books [54]