NATURAL LANGUAGE INFERENCE OVER DEPENDENCY TREES

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering

2017

By
Ali Almiman
School of Computer Science

Contents

Word Count:
40595

List of Tables

List of Figures

Abstract

NATURAL LANGUAGE INFERENCE OVER DEPENDENCY TREES
Ali Almiman
A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2017

This thesis aims to develop a Natural Language Inference (NLI) engine that is more robust and accurate than can be obtained through the current standard approaches. There are currently two main approaches to NLI: shallow and deep. Shallow approaches are based on lexical overlap, pattern matching, and distributional similarity [Giménez and Màrquez, 2007] while deep approaches employ semantic analysis, lexical and world knowledge, and logical inference [Blackburn and Bos, 2003]. Both of these approaches have advantages and disadvantages. Shallow approaches rate, as their name suggests, superficial. They cannot make use of background knowledge to link a query to a body of knowledge because there is no way of chaining through a series of rules. Deep approaches are fragile, since they can only be employed with texts that can be accurately parsed, and there are no existing parsers that can reliably analyse the structure of arbitrary input texts.

The goal here is to create an in-between approach that takes advantage of the most useful points of each of the existing approaches. The way to achieve this solution is by taking the pre-processing stage from shallow approach (dependency trees) and the inference stage from the deep approach, where an inference engine (theorem prover) has been created in a different standard. This Inference engine will obtain the required information from natural language directly, without translating inputs into any logical formula. Therefore, the goal is to apply robust logic to natural language. To achieve this goal, a theorem prover must be designed so that it can accept NL snippets. In particular, we replace the standard unification algorithm used in first-order theorem prover by an approximate algorithm for matching parse trees. We have tested this approach to NLI using rules extracted from several online dictionaries and with syllogistic patterns extracted from the FraCaS test set.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

After praise and thanks to Allah for facilitating me to complete my doctoral study at such a prestigious university, which hasn't been achieved without the favour of Allah and the support of many people who contributed to achieving this goal and completing the intended study, I express my sincere thanks and gratitude to them due to their great influence on me and the progress of my studies. On this page, I would like to dedicate to them messages of acknowledgement for what they presented at the level of research. I'd like also to thank my thesis supervisor, Professor Allan Ramsay, as he was the best supervisor to me during the research period and proficient in his specialty. He explained a lot of concepts to me and made his contributions to overcome many obstacles which faced me in the varying stages of the research period. My sincere thanks and abundant appreciation are to him. I do not forget also to thank Dr Ian Pratt-Hartmann, who has had his own imprint on guidance, explanation and comment on some points covered by this research. I also extend my thanks to Dr Bill MacCartney and Professor Christopher D. Manning from Stanford University, for providing the testing sample to the researchers clearly and smoothly. On the side of work, I'd like to thank Majma'ah University for this university scholarship and their follow-up to me and providing for all my needs without default or delay in response to my requests and inquiries.
On the family side - before everyone else - it's necessary to extend my thanks and gratitude to my virtuous parents, who taught, helped and encouraged me a lot in every phase of my personal and academic life. My father 'Abdulrahman' is the first one who taught me and imparted knowledge to me as he, may Allah preserve him, was my teacher in the first year of primary school and gave me a foundation in reading, writing, arithmetic and biology, and I still learn from his plentiful knowledge in various sciences. As for my mother 'Maryam', may Allah preserve her, she has taught me up until now a practical application of the importance of seeking knowledge and the perseverance to gain it, and she still seeks knowledge up to this moment, despite her old age and her numerous preoccupations. I would also like to thank all those who shared with me all the details of this academic experience at Manchester: my sincere wife 'Afrah' for her continuous encouragement, and coping with my preoccupation and isolation in the office for long hours, and not sharing with her the household burdens and the difficulties of the upbringing of our nice sons 'Eyad' and 'Abdulrahman'. She has carried that out to the fullest, so my bountiful gratitude and greatest appreciation are due to her. I also do not forget to offer gratitude to my colleagues and friends for their encouraging stance and their important role in providing an

attractive study environment. In particular, I thank all who remembered me and acted in my best interests, including those whose names I don't remember and those who I have accidentally omitted from this acknowledgement.

List of Abbreviations

The following table describes the significance of various abbreviations and acronyms used throughout the thesis:

Abbr.	**Full form**
BoW	Bag of words
CNF	Conjunctive normal form
CV	Consonant & vowel
DEL	Delete
DetHyps	Determiners Hypernym table
DF	Dependency framework
DIRT	Discovery of inference rules from text
DNA	Deoxyribonucleic acid
DRS	Discourse
ED	Edit distance
FOL	First-order logic
FraCaS	Framework for Computational Semantics
IC	Information content
IDF	Inverse document frequency
IE	Information extraction
INS	Insert
LCS	Longest common subsequence
LD	Levenshtein distance
LFT	Logic form transformation
LHS	Left-hand side
MP	Modus ponens
MT	Machine translation
MTGS	multiple tags

Continue in the following page ⟶

Abbr.	**Full form**
NatLog	Natural logic
NER	Named entity recognition
NLI	Natural language inference
NLU	Natural language understanding
NLP	Natural language processing
NLTK	Natural language tool kit
NNF	Negation normal from
NP	Noun phrase
NTTD	next thing to be done
OMCS	Open Mind Common Sense
POS	part-of-speech
POST	part-of-speech tag
PSF	Phrase-structure framework
RHS	Right-hand side
RTE	Recognising Textual Entailment
RTED	Robust Tree Edit Distance
SUB	Substitute
TE	Textual entailment
TED	Tree Edit Distance
TexTail	Textually entail
TexTailment	Textual entailment
TexTails	Textually entails
TF	Term frequency
TMDC	The Macmillan Dictionary
UDT	Universal Dependency Treebank
UNK	Unknown
VC	Vowel & consonant
VP	Verb phrase
WDFS	WordNet Definitions
WRTBD	What remains to be done

Chapter 1

Introduction

1.1 Overview

For any two given snippets, if one of them follows from (or is deducible from) the other, we can say that there is an entailment relation that holds between these two sentences. In the field of natural language processing (NLP), researchers interpret the definition of entailment in a slightly different way. In NLP, where there are two sentences, A and B, we can say A entails B if B is proved by A.

To clarify this notion, consider the following examples:

(1) Entailment example:

a. *He saw two black Manx cats on the doorstep.*

b. *He saw two cats.*

(2) Non-entailment example:

a. *This is Simon's car.*

b. *This is Simon's red car.*

Example (1) contains a valid entailment, since an ordinary person, upon hearing 1-a, would likely accept that 1-b follows / is inferred from 1-a. On the other hand, Example (2), is not considered to have a valid inference, as 1-b does not naturally follow 1-a. More examples can be found in the Recognising Textual Entailment (RTE) set by Dagan et al. [2006], where there are collections of problems and each problem has a text t and a hypothesis h and the answer of this problem is either 'True' or 'False'.
There are two well-known techniques that logicians and semanticists employ to check whether an entailment holds among given sentences. The first is the *logical-based approach.* The second

is the *lexical semantic approach*. The use of these two approaches represents a challenge in this field, as discussed in the following section.

1.2 Challenges in natural language understanding

As mentioned, one of the well-known techniques that logicians and semanticists employ to carry out entailment checking is the logical-based approach. To use the logical-based approach, both of the text fragments must be translated into a formal meaning representation (e.g., first-order logic) before applying automated reasoning tools that determine the inferential validity. This approach leads to a very good precision score, based on its ability to deal with several issues. Issues addressed by this approach are negations, quantifiers, and conditionals, amongst others. However, although it shows success in restricted domains, this deep approach fails in the open-domain evaluations of natural language inference (NLI). The problem lies in the core of its task, and, as such an approach, tends to flounder with the difficulty of accurately translating complex natural language sentences into logical forms.

It is more likely for logical-based approach systems—discussed in Section 2.2—to have issues when the input sentences include: idioms, intentionality and propositional attitudes, modalities, temporal and causal relations, certain quantifiers [MacCartney and Manning, 2007]. However, the methods for the lexical approach apply matching algorithms directly on simple surface representations (Sections 2.3.2 and 2.3.3), or convert the input sentences into syntactic trees and then apply the matching on these resulting trees (Section 2.3.4). Systems that utilise this technique depend on only some measures of lexical similarity between individual words by following different approaches. These approaches include bag of words, string edit distance and tree edit distance, which are all explained in the following chapter. This approach, however, is shallow, and, therefore, it is not suitable to be applied on textual entailment problems that require more than one step of inference. Therefore, the problem with the former approach is that it is brittle, while the problem of the latter approach is that it is shallow. Many systems were built on these two approaches, which has led to an interest in developing a hybrid of these systems that highlight the positive points of each approach. Shared tasks are used to test these kinds of systems and evaluate how they achieve different topics. One of these is the Recognising Textual Entailment (RTE) task [Dagan et al., 2006]. RTE is a textual entailment challenge that contains problems of two text fragments: a text t and a hypothesis h. The task is to conclude whether the meaning of one text is entailed (can be inferred) from the other text. Another example of a textual entailment test set is the Framework for Computational Semantics (FraCaS), which differs from RTE in that it has some problems with multiple premises. In this thesis, we are explaining a system that can be applied on multi-steps of inference, therefore, we chose the FraCaS test suite.

1.3 The FraCaS test suite

The FraCaS test suite of natural language inference (NLI) problems is presented as one of the projects of the FraCaS Consortium [Cooper et al., 1994], which is a collaboration that aimed to enrich the content of computational semantics. This test suite contains 346 NLI problems, and each problem has one or more premises (i.e., *P1, P2, P3, ...*) and is most likely followed by a question *q*, and an answer. A considerable amount of work was done by MacCartney and Manning [2007] to introduce another member, *hypothesis*, to FraCaS sentences, and they explain their contribution to add the element *h* in the following quote:

> *The original problems included questions, not hypotheses. But in my work, I prefer to work with declarative hypotheses. I've therefore introduced the 'h' element, containing a sentence which is, as nearly as possible, the declarative equivalent to the question posed in the 'q' element. The transformation from question form to declarative form was initially performed automatically, but all the results were manually reviewed and edited for correctness. Note that in a few questions (specifically, those containing 'any', 'anyone', or 'either'), it was necessary to change some key words to achieve grammaticality. 'Any' and 'anyone' were replaced by 'some' and 'someone', while 'either' was replaced by 'one of the'. Problems 38, 46, 54, 55, 62, 70, 71, 78, 107, 108, 141, 344 were affected in this way.*[1]

In general, the sentences that have been used in FraCaS problems are comparatively simple, and each problem has quite similar elements (i.e., *premises, questions* and *hypothesis*). Nonetheless, these problems are carefully designed to cover multiple issues in the field of NLI; the covered topics with the quantities of their problems are presented in Figure 1.1.

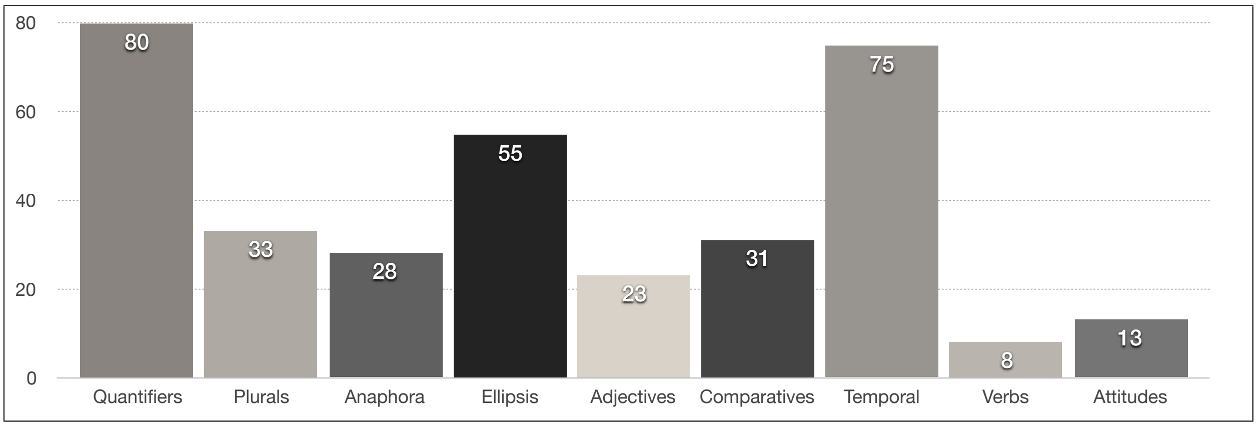

Figure 1.1: The topics covered in the FraCaS test suite

[1]http://nlp.stanford.edu/ wcmac/downloads/fracas.xml

Sec	Topic	No.	Sentences	Answer
1	Quantifiers	46	p: *Neither commissioner spends time at home.*	NO
			h: *One of the commissioners spends a lot of time at home.*	
2	Plurals	81	p: *Smith, Jones and Anderson signed the contract.*	YES
			h: *Jones signed the contract.*	
3	Anaphora	114	p: *Mary used her workstation.*	YES
			h: *Mary's workstation was used*	
4	Ellipses	177	p: *John said that Mary wrote a report, and that Bill did too.*	UNK
			h: *Bill said Mary wrote a report.*	
5	Adjectives	197	p: *John has a genuine diamond.*	YES
			h: *John has a diamond.*	
6	Comparatives	229	p: *The PC-6082 is as fast as the ITEL-XZ.*	NO
			h: *The PC-6082 is slower than the ITEL-XZ.*	
7	Temporal references	275	p: *Smith left the meeting before he lost his temper.*	UNK
			h: *Smith lost his temper.*	
8	Verbs	326	p: *ITEL built MTALK in 1993.*	YES
			h: *ITEL finished MTALK in 1993.*	
9	Attitudes	339	p: *It is false that ITEL won the contract in 1992.*	NO
			h: *ITEL won the contract in 1992.*	

Table 1.1: Random single-premise inference problems from the FraCaS test suite

Regarding the answers in FraCaS problems, each problem has one of three possible answers (i.e., 'yes', 'no', and 'unknown'). The problems with the answer 'yes' represent about 59% of the total problems, 28% are answered as 'unknown', and only 10% of the problems have the negative answer 'no'. MacCartney [2009] explained these answers, as follows:

(a) YES: means that the hypothesis can be inferred from the premise(s).

(b) NO: means that the hypothesis contradicts the premise(s).

(c) UNK: means that the hypothesis is compatible with (but not inferable from) the premise(s).

As mentioned earlier, there are single and multiple premise problems. Single-premise problems represent 55.5% of the total problems. The rest of the problems are distributed between two premises (35.3%), three premises (8.4%) and nominal values for four or five premise answers (0.8%). For some examples of single-premise problems, consider the random selection examples in Table 1.1, while the examples in Table 1.2 pertain to two-premise problems.

Sec	Topic	No.	Sentences	Answer
1	Quantifiers	13	p1: Both leading tenors are excellent.	YES
			p2: Leading tenors who are excellent are indispensable.	
			h: Both leading tenors are indispensable.	
2	Anaphora	119	p1: No student used her workstation.	NO
			p2: Mary is a student.	
			h: Mary used a workstation.	

Table 1.2: A couple of examples for multi-premise problems in the FraCaS test suite

1.4 Research goals

Multiple FraCaS premise problems are considered complex tasks, as they require deep language understanding. Generally speaking, entailment systems that sidestep translating input sentences into logical formula will be limited to handling single-premise problems only, as in the NatLog system by [MacCartney and Manning, 2007]; [MacCartney and Manning, 2008]. This is simply because all of the shallow techniques are applying one step of inference as they only compare the two sentences parts. However, in the last decade, there has been an interest in making some deep systems more robust by employing some shallow techniques. These are explained in the following chapter. This system tries to take a conversion action, where it tries to make shallow approaches less shallow by doing the following:

1. Making a less shallow system by designing a theorem prover that is able to produce proofs on trees that are generated from syntactic parsers.
2. Enabling the theorem prover to apply approximate matching between the trees' nodes, for instance, using subsumptions, skipping modifiers, and dealing with negations and quantifiers.
3. Providing background knowledge written in natural language so that it can be utilised by our inference system.

1.5 The system architecture

The proving system runs a three-stage process, as can be seen in Figure 1.2. Each of these stages will be explained in the following lines:

Normalising input text: This part of the system represents the pre-processing stage, which is responsible for converting both input *premise(s)* and a *hypothesis* from raw text forms into dependency trees. To accomplish this goal, we have used a version of MaltParser by Nivre et al. [2006] that is trained on the Penn Treebank.

Theorem proving: This part of the engine tries to extract the entailment between normalised *premise(s)* and a *hypothesis* by using some inference rules, with the aid of some textual entailing techniques. This is explained by the following lines:

- **Inference rules**: There are two types of inference rules that are employed by this prover:
  1. **Definitional rules**: These rules are extracted for this system from two dictionaries, i.e., The Macmillan Dictionary (TMDC) and WordNet definitions (WDFS).
  2. **Syllogistic rules**: These rules are extracted from double-premise FraCaS examples.
- **Textual entailment techniques**: There are three main moves that we use in this section:
  1. **WordNet hypernyms**: When comparing two different words (w_1, w_2), we check if they are related to each other, using the feature of WordNet hypernymy relations [Miller and Fellbaum, 1998].
  2. **Determiners hypernyms**: These relations are collected from different sources in Chapter 4, to compare between different generalised quantifiers.
  3. **Dropping modifiers**: These are handwritten rules that allow one to drop adjective and prepositional modifiers when applying the process of approximate matching.

Finally, the theorem prover makes the final entailment decision as to whether the entailment holds or not.

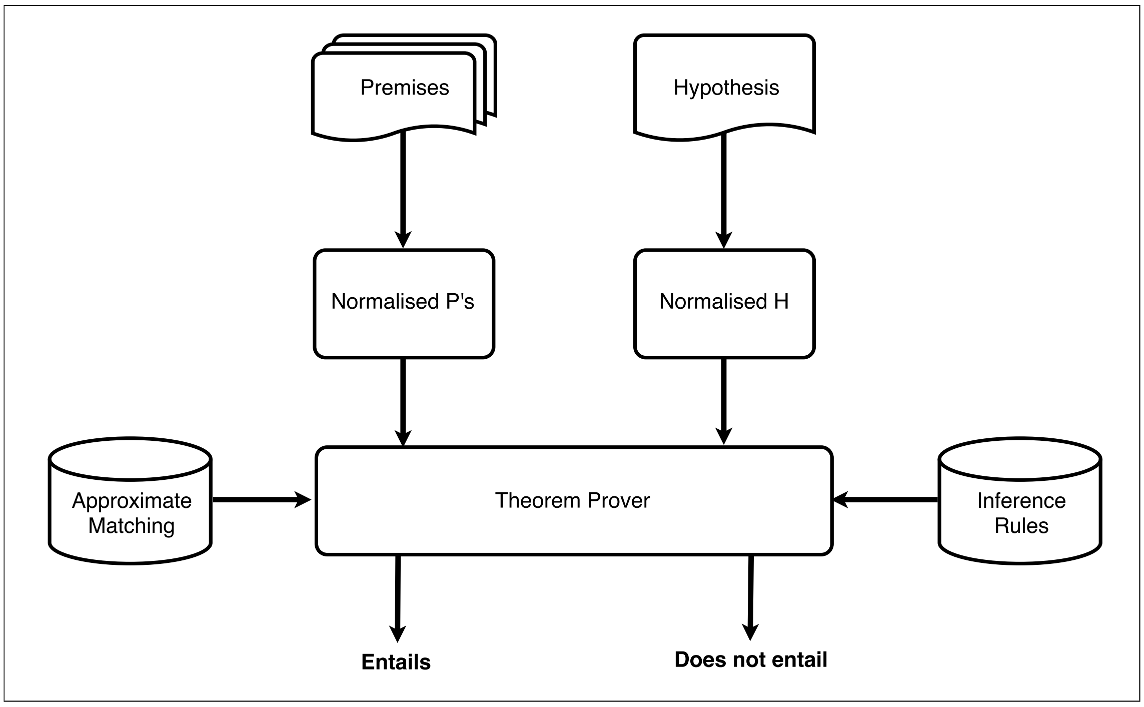

Figure 1.2: General diagram of the proving system architecture

1.6 Research contributions

The main contributions of this work are illustrated in the following points:

1. Building an inference engine that accepts inputs in a readable format (dependency trees). The current system can find entailment between two trees using inference rules. The proof system can handle Horn clause rules [Horn, 1951] where there are one or more antecedents in the right-hand side (RHS) of a rule with only one consequence in the left-hand side (LHS) of that rule. The algorithm of inference that is used in this work is a backward chaining algorithm and it can be applied for multi-step inference tasks. Besides using inference rules, this engine uses approximate matching techniques in its unification function, e.g., word subsumption and deleting modifiers. Because it is allowed to skip modifiers, the unification algorithm is considered to be a non-deterministic algorithm.

2. Creating commonsense background knowledge that fits the needs of the inference system. The rules were extracted from dictionaries, where we found definitional texts in the form “word: definition”, we changed into universal rules that contain variables. These rules were generated from two online dictionaries: The Macmillan Dictionary (TMDC) and WordNet Definitions (WFDS). In addition to these rules, some syllogistic rules have been

extracted from FraCaS double-premise problems and can be used to solve approximate problems in FraCaS.

1.7 Thesis outline

In this chapter, *Introduction*, we have presented the research problem and listed some challenges that linguists face in the field of natural language understanding. We then laid out the research goals and contributions by representing the general framework of FraCaS. The remainder of the thesis is organised as follows.

Chapter 2, *Entailment in natural language*, is a background chapter that introduces the problem of entailment, as outlined under the perspectives of both logic and linguistics. The two main approaches to conduct entailment (i.e., *deep-but-brittle* in Section 2.2 and *shallow-but-robust* in Section 2.3) are presented with examples. This is followed by various hybrid systems that benefit from both approaches.

Chapter 3, *Dependency grammar*, presents a brief overview of two well-known grammatical frameworks: phrase-structure grammar and dependency grammar. We then discuss the rule-based and data-driven frameworks for parsing, before explaining why we have used data-driven as a parsing framework and dependency grammar as a grammatical one.

Chapter 4, *Theorem proving over dependency trees*, introduces the steps we followed to build our theorem prover, which has been built to accept dependency trees. As we avoid translating to logic, this prover is able to deal directly with any syntactic parser's output. Having input of this kind, we explain how this prover can apply approximate matching techniques instead of first-order logic (FOL) unifications. A list of shallow approach techniques are used in this prover, such as using WordNet hypernyms and skipping adjectives, and prepositional modifiers. Some relations between generalised quantifiers are also presented in a separate section (e.g., $all \subset some$ and $most \subset many$). Another issue discussed in this chapter is how our inference engine dealt with *Negations* and *Polarity*.

Chapter 5, *Generating commonsense knowledge*, describes why current commonsense knowledge is not the most appropriate for the current system. We illustrate the steps that we follow to generate our inference rules from *TMDC*, and how we use the same rule-making steps for making rules from another dictionary, i.e., *WDFS*. Syllogistic rules that are extracted from FraCaS double-premise problems are also generated and explained at the end of that chapter.

Chapter 6, *Evaluation*, shows how we evaluate our theorem prover on the FraCaS one-premise problems using only the approximate matching algorithm. We then test the FraCaS two-premise problems using the syllogistic inference rules, and then test the extended set of FraCaS test set using both the inference rules and the approximate matching algorithm.

Chapter 7, *Conclusion*, contributions and future work, concludes the final remarks of the thesis. The chapter ends by suggesting directions for future improvements and research.

Chapter 2

Entailment in natural language

2.1 What is entailment?

Entailment denotes a particular association that exists between two sentences, namely, that whenever the first is true, so is the second. It is a crucial element in the use of language: we talk to each other in order to tell each other what the world is like, and the fact that adding new information to what someone already knows will enable them to build a richer picture of the world is an important part of this.

Consider, for instance, the following pair of sentences: (A) *'I will turn 30 next month'* $\models$ (B) *'I am alive'*. Clearly A entails B, since one cannot engage in birthday celebrations if one is not living. Saying A enables the hearer to enrich their picture of the world beyond the fact that the speaker will turn 30 next month. They also know that he is alive because the fact that he is due a birthday entails the fact that he is alive.

The idea of entailment, however, is very difficult, if not impossible, to capture in a computer system. Entailment is about the possible truth and falsity of various statements, and it is hard to see how we would describe these in terms with which a computer could work. What computers can do is manipulate data structures, but trying to describe truth and falsity in terms of sets of data structures is not straightforward.

There is, however, a related notion, namely, that one sentence can be derived from another. Proof is the process of applying a set of rules that are known to be reliable to get from a starting position to a target endpoint. It is like playing a game. There are moves you are allowed to make, and the aim is to perform a finite sequence of such moves to get from where you are to where you want to be.

The relationship between these two ideas has been widely studied in the field of formal logic. If they were, in fact, equivalent, then it would be possible to develop a system that could determine

whether A entailed B by determining whether A supported a proof of B.
Within formal logic, three crucial notions relating to this relationship have been investigated:

- **soundness**: a set of proof rules is said to be **sound** if whenever you use it to prove that A follows from B, then B does entail A. This seems to be a very desirable characteristic of a proof system—you really would not be able to prove things that were not true.
- **completeness**: a set of proof rules is **complete** if it could be used to derive A from B if B entails A. This again seems desirable, but is not always achievable. Some types of logic, such as propositional logic and first-order logic, have complete proof theories; others, such as higher-order logic, do not.
- **decidability**: a set of proof rules is **decidable** if it is possible to write a program which will tell you whether there is a proof of A from B for any arbitrary sentences A and B. Yet again, this seems desirable, particularly in a context where we are trying to develop computational systems, but again it is not always achievable. Propositional logic is decidable, first-order logic is not.

These relations will be important in the discussion below. We are interested in entailment: if I say B, does that mean that A is true? But it is easier to write programs that carry out proofs than to even imagine how a program could determine entailment. We will return to these issues below.

2.2 Logic-based approaches to entailment in NLP

2.2.1 Overview

In this thesis, therefore, we are interested in determining entailment relationships between sentences of natural language. One way of finding such relationships would be to translate from natural language texts into some formal logic, and then use an inference engine to determine whether one sentence can be proved on the basis of another. If the chosen logic were first-order logic, then showing whether one such logical translation, A, could be proved on the basis of another, B, would indeed show whether B entailed A, since we noted above, entailment and provability are equivalent for this logic. For this reason, approaches that translate from natural language into logic, particularly first-order logic (FOL), have been widely explored.
For instance, the following examples show typical translations from language into logic (these examples were obtained from the PARASITE system [Ramsay, 1999]). Other systems, e.g., Boxer Bos [2008], produce similar translations.

(2.1) a. *Paul and Janet are divorced.*

```
exists(_A,
    (event(_A, divorce)
    (& (theta(_A,
        object,
        (ref(lambda(_B, named(_B, 'Janet')))
        & (ref(lambda(_C, named(_C, 'Paul')))
    (& aspect(ref(lambda(_D, past(now, _D))), simplePast,_A)))))
```

b. *They had been married.*

```
exists(_A ::  past (ref(lambda (_B, past(now, _B))), _A),
    (exists(_C,
        (event (_C, marry)
        & theta(_C,
            object,
            (ref(lambda(_D, centered(_D, lambda(_E, thing(_E)))))
        (& aspect(_A, simplePast, _C)))))
```

It has, however, proved extremely difficult to scale up this kind of approach. The standard way to construct translations into logic is by annotating the rules of a grammar with semantic descriptions, and then to parse the input text using this grammar. To do this, you have to be able to write a grammar that covers the texts you want to analyse, you have to write the semantic annotations, and you have to have a parsing engine which can apply this grammar reasonably quickly to the texts that you want to analyse and which will produce exactly one analysis for any given text.

Each part of this is a major challenge. Writing wide-coverage grammars is hard, annotating their rules with semantic annotations that will produce coherent and consistent logical translations is hard and writing parsers that can make use of such rulesets to produce unique analyses of arbitrary input texts is hard. People working within this tradition have therefore often taken approaches which, while aiming to make use of the general idea, produce translations that are not guaranteed to be perfect translations.

In Moldovan and Rus [2001], for instance, the authors used open-class words; i.e., nouns, verbs, adjectives and adverbs, and turned them into predicates, where the name of the predicate is the morpheme's base form of that word. The predicates then vary, depending on the word's tag. For nouns, the predicates have a variable as an argument; e.g., $noun(x)$. Verbs are quite different, as each predicate has at least a couple of arguments. For *'intransitive'* verbs, there are two variables: (i) for the event and (ii) for the subject; i.e., $verb(e, x_1)$. As *'transitive'* verbs have objects, then they added one more variable to construct transitive verbs' predicates; for sentences, the logic form transformation (LFT) for the sentence *'a person who backs a politician'* is: `[person:n(`x_1`)`

$\wedge$ `back:v`(e_1, x_1, x_2) $\wedge$ `politician:n`(x_2)]. In the case of *'ditransitive'* verbs, there are three variables; for example, *'teacher gives pupil the grades'* is transformed to: [`teacher:n`(x_1) $\wedge$ `give:v`(e_1, x_1, x_2, x_3) $\wedge$ `grade:n`(x_2) $\wedge$ `student:n`(x_3)]. The variables in verbs' predicates are written in one order, where the event comes first, then the subject followed by the objects, if they exist, the direct object, then the indirect object, respectively. The adjectives and adverbs are following what they modify and they are added to the predicate label; e.g., *'hand-made object'* transformed to [`object:n`(x_1) $\wedge$ `hand-made:a`(x_1)] and *'walk slowly'* transformed to [`walk:v`(e_1, x_1, x_2) $\wedge$ `slowly:r`(e_1)]. Conjunctions are also transformed into predicates that aggregate some predicates as sentences, like *'an achievement demonstrating great skill or mastery'* transformed to `achievement:n`(x_1) $\wedge$ `demonstrate:v`(e_1, x_1, x_2) $\wedge$ `or` (x_2, x_3, x_4) $\wedge$ `skill:n`(x_3) $\wedge$ `great:a`(x_3) $\wedge$ `mastery:n`(x_4). For every preposition, there is also a predicate with two arguments, where the first is for the prepositional phrase and the second for the prepositional object; for instance, the LFT for *'deprive of value for payment'* is `deprive:v`(e_1, x_1, x_2) $\wedge$ `of`(e_1, x_3) $\wedge$ `value:n`(x_3) $\wedge$ `for`(x_3, x_4) $\wedge$ `payment:n`(x_4). All of these transformations are relying on some rules that are depending on the syntactic parse of a sentence. Other than the part-of-speech tag (POS), Moldovan and Rus's LFT algorithm is also designed to capture the named entity recognition feature (NER). In Tatu and Moldovan [2005], for instance, the authors employ the LFT algorithm for input sentences, and they gave an example of the sentence *'George and his relative, Mike, came to America'*, which has the following LFT: `George`(x_1) $\wedge$ `relative`(x_2) $\wedge$ `Mike`(x_3) $\wedge$ `ISA`(x_3, x_2) $\wedge$ `KIN`(x_1, x_3) $\wedge$ `came`(e_1) $\wedge$ `AGT`(x_1, e_1) $\wedge$ `AGT`(x_2, e_1) $\wedge$ `America`(x_4) $\wedge$ `LOC`(e_1, x_2).

Using an algorithm such as LFT and then applying its output into a semantic parser to get the relations, such as the work done by Tatu and Moldovan [2005] may lead to better semantics. However, according to Tatu and Moldovan, even state-of-the-art semantic parsers are not able to get all the semantic relations that the text conveyed.

2.2.2 Proof theory and automated reasoning

When the step of meaning representation is accomplished, logical-based systems become ready for the next move of checking inferential validity by doing proofs. A proof in FOL is a finite sequence of applications of proof rules that leads from a set of assumptions to a conclusion. A very popular example of proof rules is the *Modus Ponens*[1] rule (MP) [McGee, 1985], which states that when there is a conjunction of an implication *and* the hypothesis (i.e., antecedent) of the implication: $p \Rightarrow q$ together with p, we are justified to conclude that q. This is best

[1]Latin for "mode that affirms"

expressed in Equation 2.2 as follows:

$$\begin{array}{l} p \Rightarrow q \\ \underline{p \qquad} \\ q \end{array} \tag{2.2}$$

Let us consider the following PARASITE rule that can be used to prove the sentence (a) in 2.1 from the sentence (b).

(2.3) */**** MEANING 78: X and Y were married if X and Y are divorced*

```
exists(_A,
    (event(_A, divorce)
    (& (theta(_A,
        object,
        (ref(lambda(_B, named(_B, X)))
        & (ref(lambda(_C, named(_C, Y)))
    (& aspect(now, simplePast, _A)))))
⇒
exists(_A,
    (event(_A, marry)
    (& (theta(_A,
        object,
        (ref(lambda(_B, named(_B, X)))
        & (ref(lambda(_C, named(_C, Y)))
    (& aspect(ref(lambda(_D, past(now, _D))), simplePast, _A))))
```

Example 2.3 shows an inference rule with two sides of the form $LHS \Rightarrow RHS$. Using the notion of the MP proof rule, the RHS is obtainable if the LHS can be proved. In the example 2.1, there is a fact that matches the LHS; therefore, it is considered to be true, and hence, the RHS is provable hypothesis. This kind of proof, i.e., MP, can be applied repeatedly, which means that if there is no rule that can be used to deduce the proof between the hypothesis and one of the facts, then one can explore ways that guide to the desired conclusion. For instance, if the hypothesis is *'Paul proposed to Janet'*, and there is a rule that says: `(X and Y were married ⇒ X proposed to Y)`, while the LHS does not exist in the facts, then we find a rule that leads to the LHS, and when it can be proved, then the LHS is considered as a fact, as it has been proved to be true. After that, another round of applying MP leads to the wanted hypothesis.

Making a computer program that looks for the appropriate proof and then moves to a following step is done by what is called *automated reasoning*. In the literature, there is a variety of methods to do automated reasoning; however, we will discuss Tableaux and Resolution methods, as they tend to be used in most of the sophisticated theorem provers [Blackburn and Bos, 2005].

2.2.2.1 Tableaux

A tableau proof is a systematic check that makes use only of systematic concepts, which lets us know whether or not it is possible to build a model in which some given formula is true or false. When saying a formula is valid, that means 'true in all models', so a formula is valid if and only if it is not possible to falsify it in any model. Hence, a formula ϕ would be valid if and only if the systematic methods tell us that there is no way to build a model that falsified ϕ. For this reason, the tableau method is often called a refutation proof method: we show that ϕ is valid by showing that all attempts to falsify it have failed. There are a number of tableau expansions that are guided by rules regarding how to make complex formulas true (or false) by breaking them into their component formulas and giving the components the appropriate truth value. First-order tableaux have 12 rules, which are as follows:

- Negation rules: $\frac{T\neg\phi}{F\phi}$, $\frac{F\neg\phi}{T\phi}$
- Conjunctive rules: $\frac{T(\phi\wedge\psi)}{\begin{array}{c}T\phi\\T\psi\end{array}}$, $\frac{F(\phi\vee\psi)}{\begin{array}{c}F\phi\\F\psi\end{array}}$, $\frac{F(\phi\Rightarrow\psi)}{\begin{array}{c}T\phi\\F\psi\end{array}}$
- Disjunctive rules: $\frac{F(\phi\wedge\psi)}{F\phi|F\psi}$, $\frac{T(\phi\vee\psi)}{T\phi|T\psi}$, $\frac{T(\phi\Rightarrow\psi)}{F\phi|T\psi}$
- Quantifier rules: $\frac{T\forall x\phi}{T\phi(\tau)}$, $\frac{F\exists x\phi}{F\phi(\tau)}$, $\frac{F\forall x\phi}{F\phi(\tau)}$, $\frac{T\exists x\phi}{T\phi(\tau)}$

For example, suppose that we have the following world knowledge:

```
Father(Simon, John)

Alive(Simon)

∀x∀y Father(x, y) ⇒ Parent(x, y)

∀x∀y (Parent(x, y) ∧ Alive(x)) ⇒ Older(x, y)
```

we can prove the goal `Older(Simon, John)` using the tableau method (as seen in Figure 2.1). The tableau method is based on refutation, thus the first step to proving a formula is to negate it. However, in this example, the formula is True, and to negate it, we should make it False as we have done in the first line. Following the rules of tableau, all the branches that we have are closed. Therefore, the whole tableau is closed and constitutes a tableau proof for our

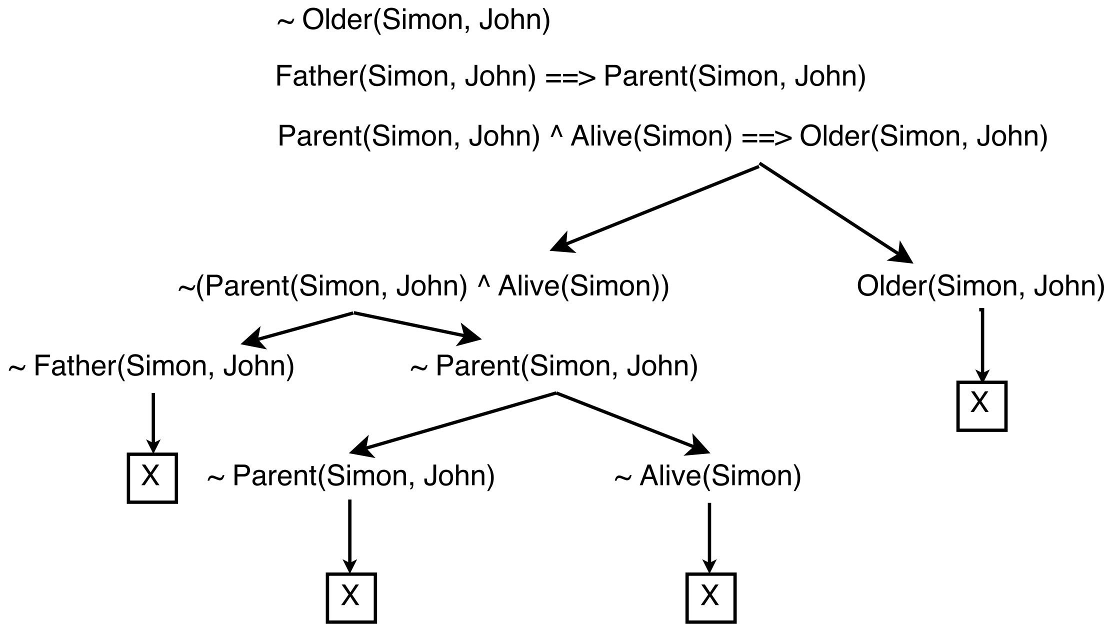
~ Older(Simon, John)
Father(Simon, John) ==> Parent(Simon, John)
Parent(Simon, John) ^ Alive(Simon) ==> Older(Simon, John)
~(Parent(Simon, John) ^ Alive(Simon))
Older(Simon, John)
~ Father(Simon, John)
~ Parent(Simon, John)
X
~ Parent(Simon, John)
~ Alive(Simon)
X
X
X

Figure 2.1: Tableaux proof steps

implication. This method of proof has been used in many existent theorem provers, such as *Lean TAP* [Beckert and Posegga, 1995], *HARP* [Oppacher and Suen, 1988], and *3TAP* [Beckert et al., 1996].

2.2.2.2 Resolution

Robinson [1965] proposed a resolution system that is different from the tableau system in two important elements. In the tableau, for each connective, there are a couple of rules to use that enable the input formula to be systematically dismantled; in the resolution system, however, there is only a single rule to use (the resolution rule), which is repeatedly applied. Furthermore, tableaux can be applied directly from an input formula, while resolution formulas have to be translated into conjunctive normal form (CNF), which will be discussed in the next subsection. Therefore, the resolution system is applied in two stages: translating to CNF and applying the resolution rule phase. The principle that is used in resolution proofs is that when trying to prove a statement, assume that the negation of the statement is true and try to arrive at a contradiction. This can be explained better by practical application, starting from the idea of transforming FOL formulas into CNF.

Conjunctive normal form (CNF):

To turn an FOL sentence into its conjunctive normal form, there are three steps that should be taken. Firstly, each FOL formula is turned to its negation normal form NNF. This step contains seven rules that followed to obtain the required version of a given formula. These rules are as follows:

1. If the given formula of the form $\neg(\phi \wedge \psi)$ **rewrite it as** $\neg\phi \vee \neg\psi$
2. If the given formula of the form $\neg(\phi \vee \psi)$ **rewrite it as** $\neg\phi \wedge \neg\psi$
3. If the given formula of the form $\neg(\phi \Rightarrow \psi)$ **rewrite it as** $\phi \wedge \neg\psi$
4. If the given formula of the form $\phi \Rightarrow \psi$ **rewrite it as** $\neg\phi \vee \psi$
5. If the given formula of the form $\neg\neg\phi$ **rewrite it as** ϕ
6. If the given formula of the form $\neg\forall x\phi$ **rewrite it as** $\exists x\neg\phi$
7. If the given formula of the form $\neg\exists x\phi$ **rewrite it as** $\forall x\neg\phi$

The second step to obtain the CNF for an FOL formula is to skolemise[2] away all existential quantifiers ($\exists$). For instance, if the formula is $\forall x\exists y P(x, y)$, it can be skolemised if y is substituted by $g(x)$, so the formula $\forall x P(x, g(x))$ is skolemised. Finally, all universal quantifiers ($\forall$) need to be discarded; doing that should not affect anything in the semantics of the formula. For example, the formula $\forall x\forall y$ `Father(x, y)` $\Rightarrow$ `Parent(x, y)` can be transformed in two steps. Firstly, turn the formula into its NNF form, using the fourth rule from the list above, to get to the form $\neg\forall x\forall y$ `Father(x, y)` $\vee$ `Parent(x, y)`. Secondly, discard all the universal quantifiers to get: $\neg$ `Father(x, y)` $\vee$ `Parent(x, y)`. To turn this formula from NNF to CNF, all conjunctions between formulas and disjunctions within formulas are turned to a comma symbol (,). Therefore, the previous formula $\neg$ `Father(x, y)` $\vee$ `Parent(x, y)` is turned into `[`$\neg$ `Father(x, y), Parent(x, y)]`. When all formulas are turned into CNF, then the succeeding step is to apply the resolution phase.

The resolution phase:

The resolution phase depends on a notion of discarding complementary literals; thus, if there are two clauses, A and B, and there is a positive literal q in clause A while clause B contains its negation $\neg q$, these literals (i.e., q and $\neg q$) represent a complementary pair. For example, if the input clauses are $[p, \neg q, r]$ and $[s, q, t]$, the output (after discarding the complementary pair) will be $[p, r, s, t]$. From the example, it is clear that the resolution system is similar to tableaux because they both use refutation methods as they negate ϕ in the input and try to generate a

[2]Skolemisation is a method for removing existential quantifiers from formal logic statements.

contradiction to prove that ϕ is valid. However, resolution is different with regards to translating the input into set CNF before applying its rule. Suppose that we have these two input clauses $[P(x), Q(x)]$ and $[\neg P(a), R(z)]$. It is obvious that we do not have any complementary pairs of literals; however, there is a way of unifying (x) with (a), and the formula will be as follows: $[P(a), Q(a)]$ and $[\neg P(a), R(z)]$; by applying the resolution rule, we will get $[Q(a), R(z)]$. As we did not reach an empty clause, we could not prove this formula. As a clear example, consider the same example that we used in the tableaux (using the same knowledge base and trying to prove the same goal). Using the resolution method of proof, it can be proved (as seen in Figure 2.2).

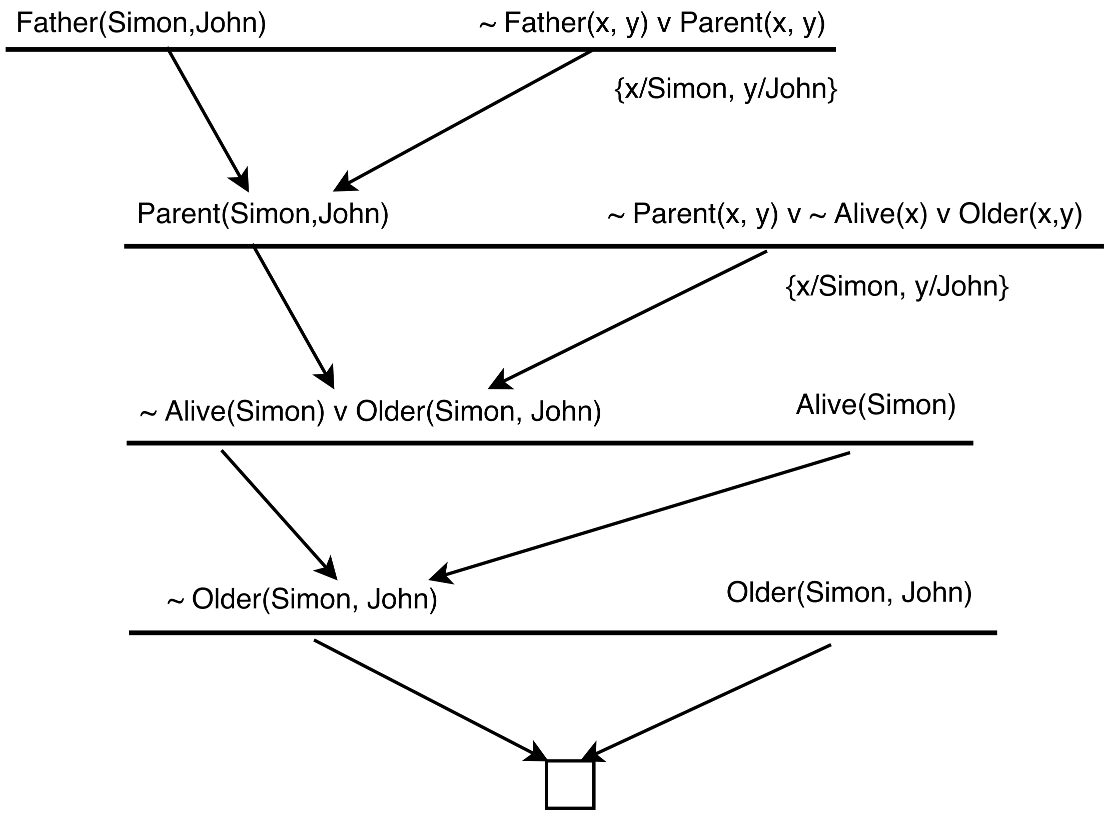

Figure 2.2: Resolution proof steps

The first step taken in Figure 2.2 is negating the hypothesis, then doing the proof shown in the figure. In the last step, there is a refutation elimination between the hypothesis with its negated version, which leads to an empty list, and that is how the resolution method works. This method of proof has been used in many existent theorem provers, such as *Vampire* [Riazanov and Voronkov, 1999][3], *Spass* [Weidenbach, 1999][4], *Superposition* [Bachmair and Ganzinger,

[3]http://www.vprover.org

[4]http://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/

1994], *FDPLL* [Baumgartner, 2000][5], *Gandalf* [Tammet, 1997][6], *Otter* [McCune, 1989][7], *Bliksem* [Ganzinger and De Nivelle, 1999][8], *Scott* [Slaney, 1993][9] and *SNARK*[Stickel et al., 1994]. More recent systems that apply this kind of proving system include those developed by Fowler et al. [2005], Raina et al. [2005] and McCune [2010].
A similar approach to resolution was utilised in our theorem proving system. The main difference between these approaches is that our system is working with non-logical formulae, as discussed in Chapter 4. The world knowledge that was used in the previous example was built just to solve this example. However, inference systems require a large amount of this kind of knowledge. There is more about this subject in the following subsection.

2.2.3 Commonsense knowledge

One of the benefits of the deep approach is that it enables you to carry out proofs involving multiple steps from what you already know about the world. For this to be of any value, you have to indeed know something about the world, i.e., to have a body of *'background knowledge'* or *'commonsense knowledge'*. Broadly speaking, the ultimate goal for artificial intelligence is to make smart machines that can think (as human beings do) [Zang et al., 2013]. That requires developing machines that are able to deal with problems and solve them just as humans would, which is one of the main targets in this field. People know some simple facts from their daily lives that often guide their attitudes either for or against something; for instance, my six-year-old son will not touch boiled water, as he knows that his finger will burn if he puts it in boiled water, so he acts upon this knowledge. In *The Emotion Machine*, Minsky defined commonsense in the following way:

> *Commonsense includes not only what we call commonsense knowledge - the kind of facts and concepts that most of us know - but also the commonsense reasoning skills which people use for applying their knowledge* [Minsky, 2006].

In the last two or three decades, this problem in artificial intelligence gained importance because automatic reasoning systems are used to decide whether there is an inference between any given sentences or not. With the lack of this knowledge, inference systems would not be able to infer the relationship between sentences that contain lexical relations between their nodes, even in simple sentences; e.g., *'I saw the sunlight'* and *'I saw the light from the sun'*, as they simply are not smart enough to naturally derive this relationship between the objects in both of the given sentences. Magnini et al. [2014b] addressed this issue when they said: *"Knowledge resources are crucial to recognise cases where T and H use different textual expressions (words, phrases)*

[5]http://www.uni-koblenz.de/ peter/FDBLL/
[6]http://deepthought.ttu.ee/it/gandalf/
[7]http://www.mcs.anl.gov/research/projects/AR/otter/
[8]http://www.ii.uni.wroc.pl/ nivelle/software/bliksem/
[9]http://users.cecs.anu.edu.au/ jks/scott/

while preserving entailment".

As a consequence, there was an interest in building databases that contained such low-level information to be used in natural language inference applications. Nowadays, there are some projects for this purpose, such as OpenCyc[10] [Lenat, 1995], ThoughtTreasure [Mueller, 1998], Open Mind Common Sense (OMCS)[11] [Singh et al., 2002], ConceptNet[12] [Liu and Singh, 2004] and YAGO [Suchanek et al., 2007]. Most of these examples are written with some semantic annotations, as they are only used in deep inference system engines (i.e., theorem provers) that are designed to only accept knowledge of this kind. For instance, in YAGO, the fact that Einstein won the Nobel Prize in 1921 is stored as `won-prize-in-year(Einstein, Nobel-Prize, 1921)`. Similarly, the sentence *'George W Bush is a person'* is written in Cyc as `(#$isa#$GeorgeWBush#$Person)`.

OMCS has different types of knowledge, as it contains raw English sentences that carry relationships between objects or events. These statements are written in simple phrases, such as *'A coat is used for keeping warm'*, *'The sun is very hot'* and *'The last thing you do when you cook dinner is wash your dishes'*. It also contains information on the emotional content of situations, in such statements as *'Spending time with friends brings happiness'* and *'Getting into a car wreck makes one angry'*. In addition, OMCS contains information on people's desires and goals, both large and small, such as *'People want to be respected'* and *'People want good coffee'* [Speer et al., 2008].

One of the ways that OMCS and YAGO feed their knowledge base is by employing the public to contribute to such word games as 'Verbosity'. This is an online game played by two players: a 'Narrator' and a 'Guesser'. The Narrator describes a secret word to the Guesser by sending hints. The form of these hints is taken from sentence templates with blanks to be filled in. An example of these templates is: `"___ is a kind of ____"`; then the answer is semantically annotated and categorised as a hierarchical relation. Other templates also exist, such as `"___ is used for ____"`, which provides information about the purpose of a word, `"___ is typically near/in/on ____"`, which provides spatial data and `"___ is the opposite of ____"` and `"___ is related to ____"`, which both provide data about basic relationships between words [Von Ahn et al., 2006]. For an example, consult Figure 2.3; to check if the word is LAPTOP, the Narrator might say, "It has a KEYBOARD". This game, however, is inspired by the popular game Taboo^{TM}, but one major difference between Taboo^{TM} and Verbosity is the use of sentence templates. Players in Taboo^{TM} can describe the secret word using arbitrary language, while Verbosity players are restricted to the use of only the types of sentences that are available to them at the time. One reason that Verbosity is required to use these restrictions

[10]http://opencyc.org
[11]http://media.mit.edu/research/groups/5994/open-mind-common-sense
[12]http://conceptnet5.media.mit.edu

is because some English sentences contain multiple meanings; therefore, these restrictions help to disambiguate the meanings and avoid such difficulties.

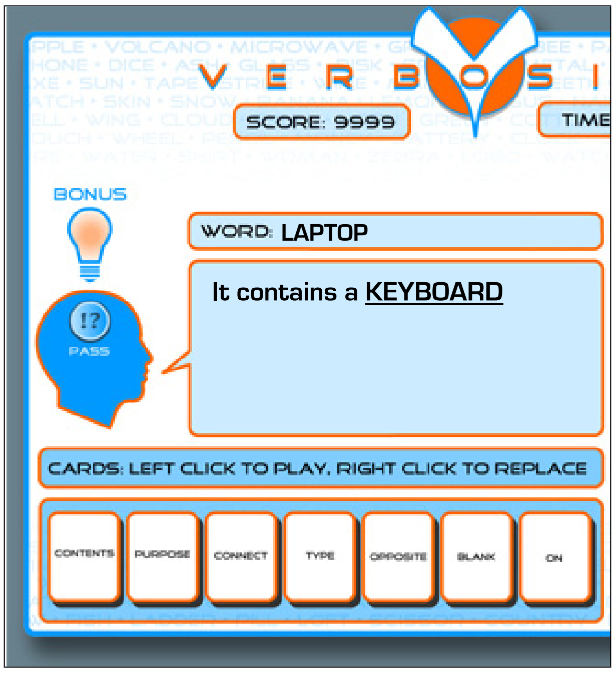

Figure 2.3: Part of the Verbosity narrator screen

OMCS data has been transferred to two kinds of machine readable representations: ConceptNet [Liu and Singh, 2004] and AnalogySpace [Speer et al., 2008]. Figure 2.4 shows an example from ConceptNet that describes the commonsense relationships between some natural language words and phrases in nodes and links surrounding the concept "cake".

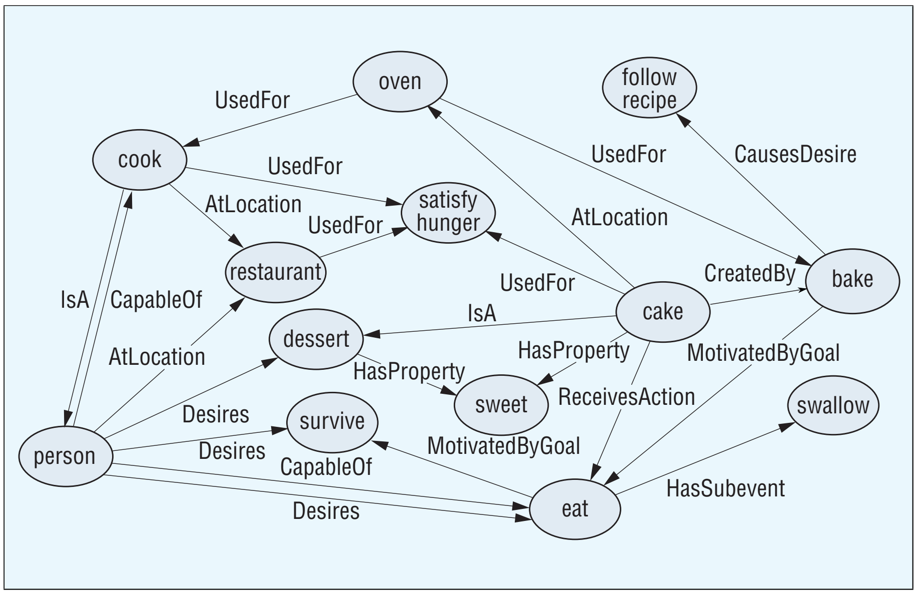

Figure 2.4: Some nodes and links surrounding the concept 'cake' in ConceptNet

Lin and Pantel [2001] devised a different solution to discover inference rules by employing an unsupervised algorithm that is considered to be an extension to the distributional hypothesis that assumes all words that occur in the same contexts are tending to be similar. However, in this extension, the similarity is applied on dependency paths rather than single words. To achieve this, for each given dependency parsed sentence, they extracted a couple of content words for the given tree, and replaced them with variables. The content words must be nouns, and the relations that are considered are only the ones that are held between content words (e.g., nouns, verbs, adjectives and adverbs). For instance, consider this example in Figure 2.5 that is taken from their paper.

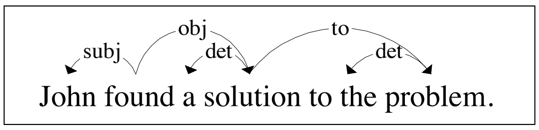

Figure 2.5: DIRT dependency path by Lin and Pantel [2001]

From Figure 2.5, their algorithm would ignore the relation between *'a'* and *'solution'* because *'a'* is not a content word (similarly for *'the'* with *'problem'*), thus, the extracted paths would be something like:

- N:subj:V ← find → V:to:N
  ≡ *X finds something to Y*
- N:subj:V ← find → V:obj:N →to:problem
  ≡ *X finds Y to problem*
- N:subj ← John:V ← find → V:obj:N →to:N
  ≡ *John finds X to Y*
- N:subj:V ← find → V:obj:N
  ≡ *X finds Y*
- N:subj:V ← find → V:obj:N → solution → N:to:N
  ≡ *X finds a solution to Y*

When such patterns as these are extracted, then the extended distributional hypothesis is applied, which is to consider the meanings of every couple of paths with a similar context to be similar. Therefore, from the above examples, they have checked a newspaper corpus to look for similar contexts for these paths and they found that the nearest path was between: *'X finds a solution to Y'* with *'X solves Y'*. Thus, they can extract an inference rule that says:
'X solves Y ≈ X finds a solution to Y'. Padó and Dagan [2016] discussed this class of entailment when they said, *"This class is composed of transformations that reflect general choices in the surface of realisation while keeping the lexical elements and the semantic relationships*

between them constant". The method that Lin and Pantel [2001] followed to evaluate their work is by parsing 1GB of newspaper text, which let them extract 231,000 unique paths. They have extracted several suggested paths for sentences that are extracted from the top questions in the Text REtrievial Conference (TREC), where some of the sentences have more than one path, while others do not have any path. Table 2.1 shows six questions with the evaluation of the top 40 most similar paths. The paths' column is the output of DIRT, where the *DIRT* column contains the number of similar constructions to the original path that DIRT has found, while the *Man.* column shows the number of the paths that are manually generated paraphrases of TREC-8 which were then compared with what the DIRT system generates in the next column, recording the intersections in the *Int.* column. The last column *Acc.* represents percentages of correctly classified tags.
Since they were extracted as raw English texts, DIRT rules are suitable to be used in shallow approach systems that do not transform their input sentences into logical forms, as in the work done by Dinu and Wang [2009]. Dinu and Wang have extended the set of DIRT rules as they found that they do not cover many of the RTE examples, such as: *X write Y* → *X author Y* and *X represent launch Y* → *X produce Y*. The extension was done by writing a hand-crafted set of rules to cover the missing rules. After this, they applied the collection of rules (both the DIRT and the extended DIRT) to tree skeletons of RTE problems, and they were able to reach 61% precision.

These examples of background resources are used frequently by inference systems. Like any other inference engine, our inference system requires background knowledge to help in the process of deducing more problems. However, these examples were not the exact form of commonsense that we are looking for. In our system, we are going to use approximate matching between raw English texts, so any data set that contains information written in some kind of logic is not suitable for our system. DIRT is one of the sources that keeps its forms written in raw English; however, what we are illustrating in Chapter 5 is somewhat different from DIRT, as we are making definitional rules.

2.3 Textual entailment (TexTailment)

2.3.1 Overview

As we have seen, obtaining logical forms and using an existing theorem prover to draw out entailment relations between natural language texts is very challenging. An alternative approach is to try to develop more approximate ways of dealing with the issue. Dagan et al. [2006] defined textual entailment as follows: *"We say that T entails H if the meaning of H can be inferred*

Q	Question	Paths	Man.	DIRT	Int.	Acc.
Q1	*Who is the author of the book, 'The Iron Lady: A Biography of Margaret Thatcher"?*	*X is author of Y*	7	21	2	52.5%
Q2	*What was the monetary value of the Nobel Peace Prize in 1989?*	*X is monetary value of Y*	6	0	0	N/A
Q3	*What does the Peugeot company manufacture?*	*X manufactures Y*	13	37	4	92.5%
Q4	*How much did Mercury spend on advertising in 1993?*	*X spend Y*	7	16	2	40.0%
		spend X on Y	8	15	3	37.5%
Q5	*What is the name of the managing director of Apricot Computer?*	*X is managing director of Y*	5	14	1	35.0%
Q6	*Why did David Koresh ask the FBI for a word processor?*	*X asks Y*	2	23	0	57.5%
		asks X for Y	2	14	0	35.0%
		X asks for Y	3	21	3	52.5%

Table 2.1: Some results of the DIRT algorithm from Lin and Pantel [2001]

from the meaning of T, as would typically be interpreted by people". That is aligned with the guidelines of the Recognising Textual Entailment (RTE)-4 challenge: *"T entails H if the truth of H can be inferred from T within the context induced by T"*.

This is an essentially proof-theoretic notion. It is about what a person might infer, not about whether something is true or not. As such, the observation in Section 2.1 that entailment and provability are equivalent in certain formalisms does not hold, and is indeed not relevant. If the interesting question is whether someone would infer the meaning of H from T, then concentrating on proofs is exactly what is required.

However, this does lead to a potential degree of confusion. Entailment, as we saw above, is about whether H is bound to be true if T is. Dagan's definition is about whether H can be derived from T by the inference strategies employed by a typical native speaker. As such, the meaning of textual entailment is not really entailment as normally understood. I therefore prefer to write it as *TexTailment* throughout the thesis, rather than repeating the terminology of two words, *textual entailment*, and *TexTail* instead of *textually entail*, partly for brevity but also to emphasise the difference between the two notions. In the following sections, well-known techniques for the TexTailment approach (bag of words, string edit distance and tree edit distance) are listed, with a brief description.

2.3.2 Bag of words (BoW)

The bag of words model, as its name would suggest, is a process of representing each document in a bag or a set of its words. It represents texts, T, and hypotheses, H, by breaking them down into multisets of their words, regardless of the word order but keeping multiplicity [Manning et al., 2008] and [Sriram et al., 2010]. When text fragments (T and H) have been tokenised and appended to lists, then there is a possibility of finding a match between their components (e.g., *T:'He is a nice man'* and *H: 'He is a nice person'*) by using WordNet relations or by finding a similarity between these words using any similarity measure. Without employing measures of lexical similarity, BoW considers the words 'tiger', 'cat' and 'house' as equally distant, despite the fact that semantically, 'tiger' should be closer to 'cat' than 'house' [Le and Mikolov, 2014]. The role of these functions is to map ordered pairs of words to real values, where the maximum equals 'one' and the minimum is 'zero'. The result of 'one' is given to equal or close words, whereas dissimilar words gain 'zero'. After applying this to all words, then the total similarity score is compared against a threshold, where this threshold is usually learned automatically from a development set of $T - H$ pairs. Some examples of using BoW to find TexTailment relations have been done by Glickman et al. [2005], Adams [2006], Adams et al. [2007], and Jijkoun et al. [2005].

2.3.3 String edit distance

String edit distance is another approach that is used in surface string similarity, the task being to find the minimum cost that is required to map the whole content of a text T into the content of a hypothesis H [Magnini et al., 2014b]. Mappings are performed as sequences of editing operations, namely: insertion, deletion and substitution. Each of these operations has a cost and these costs vary according to the method used. A popular algorithm for string edit distance is named for its proponent, Levenshtein distance (LD) [Levenshtein, 1966], is followed by some variants, such as: Hamming distance [Sankoff and Kruskal, 1983], Episode distance [Das et al., 1997] and Longest common subsequence distance [Needleman and Wunsch, 1970].
Levenshtein distance (LD) by Levenshtein [1966] is an approach that accepts three edit operations, each with a cost of 1, which are insertions, deletions and substitutions. There are many available paths to transform the first sentence into the second, but LD is looking for the minimum number of editing operations, and this method is called (minimum edit distance) and preserves the matched words. Therefore, the minimum edit distance is an algorithm that looks for the shortest path to reach its goal. The edit distance between n and m is $D(n, m)$; consider the following example:

- If n is 'This victim was assassinated' and m is 'This victim was assassinated' then *D(n,m) = 0*, because no operation is needed. The words are already identical.
- If n is 'This victim was assassinated' and m is 'This victim was killed' then *D(n,m) = 1*, because one operation (substitution) is sufficient to transform n to m.

The Levenshtein distance solves the string edit distance problem by using dynamic programming [Dasgupta et al., 2016] and [Cormen, 2009]. It takes the main problem and splits it into sub-problems, solving each sub-problem only once and remembering the results in matrix D for later. Then it computes its solution bottom up by synthesising it from smaller sub-solutions and by trying several possibilities and choices before it arrives at the optimal set of choices for the whole problem. The idea here is to use the cheapest edit operations possible to transform one string to another. Typically, each edit operation costs 1, except that exchanging similar items costs 0. So the Levenshtein distance does not calculate the similarity between two strings, but, rather, the distance in cost measures. Some variants of the LD algorithm consider variable costs, as the operation of deleting an adjective modifier is not similar to deleting a noun, and substituting synonyms differs from substituting non-related words. An example for using this technique to solve TexTailment problems is considered by MacCartney and Manning [2007] and MacCartney and Manning [2008].

2.3.4 Tree edit distance (TED)

Recently, comparison of tree-structured data has been a growing interest in several diverse areas, such as image analysis, XML databases, automatic theorem proving, computational biology and NLP [Mehdad and Magnini, 2009]. Tree edit distance (TED) is considered to be one of the most effective techniques in this field. Similar to the previous section, i.e., *string edit distance*, TED's metric is to calculate similarities between rooted ordered trees rather than simple strings. Selkow [1977] proposed an extension to the string edit operations used in the Levenshtein distance. Selkow implemented a recursive algorithm for computing the minimum sequence of operations that transform one tree to another. In Selkow's algorithm, delete and insert operations are specific operations that are enabled solely on tree leaves, whereas exchange can be applied at every node. There is a nonnegative real cost associated with each edit operation, and similar to LD, some systems apply variable costs. One of the instances of employing TED on TexTailment tasks can be found in the work of Kouylekov and Magnini [2005], where they used variable costs. The reason for using variable costs is that the words in a sentence are not equally important, so each word is assigned to its inverse document frequency (idf) weight. The term idf is widely used in Information Retrieval (IR), where it can be calculated by looking for two indicators: (i) the number of documents in a text collection N and (ii) the number of documents that contain the target word N_w. From these two figures, the idf for a word w can be obtained by applying the Equation 2.4:

$$idf(w) = log\frac{N}{N_w} \tag{2.4}$$

The idf weight of a word is its 'insertion' cost, thus most common words as stop words have a cost of zero when they are inserted.

In both of the edit distance techniques in Sections 2.3.3 and 2.3.4, TexTailing systems employ a threshold that is compared against the total cost of the sequence of edits; if the total cost is within that threshold then there is TexTailment, otherwise TexTailment does not hold.

An advantage of using these examples of TexTailment techniques is that they deal with input sentences directly without being transformed to any kind of logic. However, they are not as suitable for tasks that require multi-steps of inference, such as what automated reasoning can do.
These two problems were recognised by other researchers in this field, so they developed the idea of using the benefits of both approaches. Some ideas that are followed in making hybrid systems are mentioned in the following section.

2.4 Hybrid entailing systems

2.4.1 Overview

In the previous Sections 2.2 and 2.3, two different methods were discussed that NLP researchers follow in order to perform entailing/TexTailing tasks. It was clear that there is no model solution to these problems. In the logic-based approach (the deep-but-brittle approach), there is difficulty in representing the meaning for complicated sentences very efficiently. At the other extreme, the shallow-but-robust approach cannot do more than one step of inference. Another issue that people are trying to solve with the logic-based approach is its low recall; therefore, some actions were taken to try to make these approaches more robust, as can be seen in the following subsections.

2.4.2 Less-shallow-approach

One way of thinking about a solution to the problems of shallow and deep approaches is to try to do inference at the level of the language without translating input sentences into logic. In this thesis, we are trying to use inference by employing raw English texts. This system was partially inspired by the NatLog system [MacCartney and Manning, 2007; MacCartney and Manning, 2008]. As they stated: *"We explore a different point on the spectrum, by developing a computational model of natural logic, that is, a logic whose vehicle of inference is natural language"*. MacCartney and Manning's system is an alignment system utilising the Levenshtein string edit distance. A feature they add to normal edit distance applications is to consider the monotonicity of contexts. The monotonicity, as the authors formally defined, is split into three points as follows:

- f is **upward-monotone** (↑) iff for all $x, y \in \alpha$, $x \subseteq y$ entails $f(x) \subseteq f(y)$.
- f is **downward-monotone** (↓) iff for all $x, y \in \alpha$, $x \subseteq y$ entails $f(y) \subseteq f(x)$.
- f is **non-monotone** (↑↓) iff it is neither upward nor downward-monotone.

By employing this notion of monotonicity, MacCartney and Manning's system can handle problems that most of shallow TexTailment systems get wrong; for instance, consider Example 2.5:

(2.5) *T: Every watch is on sale until the next week.*
H: Every Swiss watch is on sale.

In Example 2.5, we can say that *H* follows from *T* because if every watch is on sale then the narrower version of the scope, i.e., 'Swiss watch' is also on sale. The other modifier, which is the prepositional modifier 'until the next week', can be dropped. In the case of upward monotone, i.e., positive context, deleting modifiers preserves truth, while in downward monotone, inserting modifiers is the action that preserves truth. Nonetheless, 'every' is considered to have downward monotonicity for its restrictor NP only. Inspired by NatLog, our system is designed to change the way of matching sentences according to the direction of the context; see Section 4.2.1.4. For substitutions, NatLog uses WordNet measures of synonymy, hyponymy and antonymy. Besides this, rules were applied on some natural language expressions to extract entailment relations, and one of the topics covered by Natlog is 'quantifiers'. Among the relations between quantifiers, the NatLog system has, for instance: (*everyone* $\subseteq$ *someone*, *all subseteq most* $\subseteq$ *some*). Relevant to this point, we extended this collection of quantifiers' relations in Section 4.2.1.3.

Bar-Haim et al. [2007] took a step further than normal matching in dependency trees when they utilised inference rules. This system was limited to simplified sentences where the hypothesis is generated from the text (not embedded in it). When two trees are given to the prover, it tries to generate *H* from *T* by applying some entailment rules, and it only works if *H* has a relatively simple structure. The main relevant part of what they do is not the process of generating *H* but the TexTailment rules that the system follows, which are quite similar to what our system is doing. These rules are actually handwritten, and they can be applied if two conditions are applied in the hypothesis, which are: a relatively simple tree and a non-negated sentence. There are two types of rules: (i) manually created and (ii) automatically obtained. The manually created rules are divided into four categories: syntactic-based, polarity-based, negations, and generic default rules. However, the automatically obtained rules are mainly from DIRT [Lin and Pantel, 2001]. For the manually created rules, it is not clear how these rules are applied, but there are some examples for what these rules do. The polarity-based rules agree that TexTailment is found for the following problem:

*John **knows** that Mary is here ⇒ Mary is here.*

*John **believes** that Mary is here ⇏ Mary is here.*

Negation rules are created to capture negated words, such as the following example:

*What we've **never** seen is actual costs come down ⇏ What we've seen is actual costs come down.*

Generic default rules are dealing with dropping modifiers, whereas syntactic-based rules have three functions: (i) simplification and canonisation of the source tree, (ii) extracting embedded propositions and (iii) inferring propositions from non-propositional subtrees of the source tree.

2.4.3 More robust deep approach

As mentioned in Section 2.2, translating to logic is one of the biggest obstacles to logic-based entailment. However, this big issue did not prevent people from trying different ways of generating better versions of meaning representations. One such attempt is the step of generating the logic form translation (LFT) for input sentences by Moldovan and Rus [2001] before applying the output into a semantic parser, such as: Tatu and Moldovan [2005] and Fowler et al. [2005]. According to Tatu and Moldovan [2005], this move allows obtaining slightly enhanced semantic outputs. For complicated sentences, it is very hard for semantic parsers to extract all of the relations from the text. Other systems, such as those developed by: Akhmatova [2005], Raina et al. [2005], and Hickl et al. [2006] began with syntactic parsers before applying syntactic output to semantic parsers.

When the syntactic tree is obtained, Raina et al. [2005] translated the relations into a form of logic, where each node in the graph is converted into a logical term. For instance, consider the following example from their paper: *'T: Bob purchased an old convertible'* and *'H: Bob bought an old car'*. The logical representation after the refutation of the hypothesis for this example is as follows:

T: `(∃ A, B, C) Bob(A) ∧ convertible(B) ∧ old(B) ∧ purchased(C, A, B)`

H: `(∃ A, B, C) ¬ Bob(X) ∧ ¬ car(Y) ∧ ¬ old(Y) ∧ ¬ bought(Z, X, Y)`

In the implication of this abductive theorem prover, there are some extended rules that are more robust than normal theorem provers. For two predicates, A and B, a unification between these two predicates holds if they are similar, or if they are different but are synonyms of each other (a similar rule is applied by Fowler et al. [2005] and Akhmatova [2005]; see WordNet feature with the theorem prover in Figure 2.6). Before applying lexical resources such as WordNet, Akhmatova [2005] splits the text T and the hypothesis H into multiple propositions. A similar action was taken by Levy et al. [2014].

The number of arguments for A and B could be different, and the unification holds when one of the two predicates contains a modifier. Two constant arguments could unify with each other.

Similarly, the first two points are applied in our existing theorem prover, but instead of using WordNet synonyms, we use WordNet hypernyms.

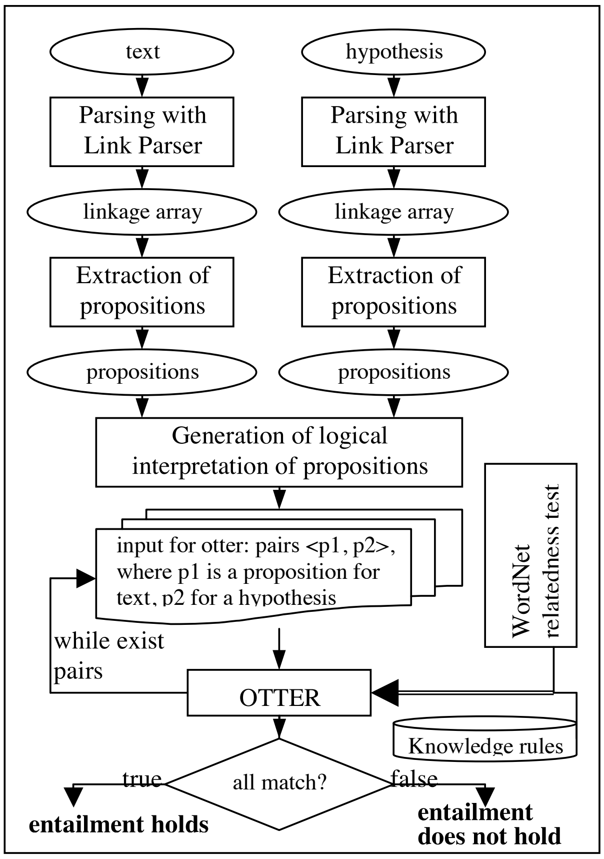

Figure 2.6: The architecture system for Akhmatova [2005]

The inference engine that we implement in our work is a combination of three algorithms, as follows:

1. **Proving algorithm:** This theorem prover is made so that it can accept dependency trees as input sentences. It is different from the way that Bar-Haim et al. [2007] follow, as they are applying the rules on the level of trees but their system is not applied on multi-steps of inference. It is also different from Raina et al. [2005] and Akhmatova [2005] as its sentences are raw English sentences.

2. **Inference rules:** This is a differently behaved theorem prover, and the rules that it can deal with would be more convenient if they were written without semantic annotations, or any logical form. Therefore, some of the commonsense knowledge was not really appropriate to such a system. DIRT is written without unusual-looking symbols; however, the kind

of knowledge that we prefer is definitional rules. Consequently, we extracted definitional inference rules from an online dictionary, as can be seen in Chapter 5.

3. **Approximate matching algorithm:** a non-deterministic unification algorithm that applies some approximate matching rules at tree node level. These techniques involve: dropping (adjective/prepositional) modifiers, dealing with negation, utilising polarity, similar to Bar-Haim et al. [2007], and using WordNet hypernyms and antonyms, similar to the algorithms that were developed by Raina et al. [2005] and Akhmatova [2005]. Inspired by MacCartney and Manning [2007], we make our system able to change the direction of unification according to the direction of the context.

Authors	Approach's task	Logic?	RTE	
			V	acc
Akhmatova [2005]	Splits the text T and the hypothesis H into multiple propositions and attempts to validate each proposition in H using a logic prover.	Y	1	0.51
Raina et al. [2005]	Uses graphs for knowledge representation and employs graph matching algorithm to perform logical inference.	Y	1	0.57
Fowler et al. [2005]	Generates axioms as linguistic rewriting rules that are used in COGEX logic prover.	Y	1	0.55
Bos and Markert [2005]	Complements theorem proving with model building as an alternative way of drawing logical inferences.	Y	1	0.61
Hickl et al. [2006]	Employs an entailment classifier that relies on four features: alignment, dependency, paraphrase and semantic.	N	1	0.65
Bos and Markert [2006]	Uses a hybrid system that combines deep features from a theorem prover and a model builder, together with shallow features as lexical overlap.	Y	2	0.60
[MacCartney and Manning, 2007]	Uses natural logic to calculate inference relations between two superfly aligned sentences.	N	3	0.63
Bar-Haim et al. [2007]	Formulates the inference problem as analogous to proof search, using inferential rules which encode knowledge of lexical relatedness.	Y	NA	NA
[MacCartney and Manning, 2008]	Extends MacCartney and Manning, 2007 by making the system cover other topics, such as: semantic exclusion and implicativity.	N	3	0.59
Clark and Harrison [2008]	Uses a logical-based system utilising various resources including WordNet and DIRT paraphrases and is tolerant, in some degree, to partially unproven H sentences.	Y	4	0.65
Dinu and Wang [2009]	Extends a version of DIRT rules with synonyms from WordNet.	N	3	0.61
Tian et al. [2014]	Computes logical relations among the corres- -ponding abstract denotations.	Y	4	0.59

Table 2.2: Some related systems that tested on RTE

Most of the examples mentioned in this chapter are allocated in the Table 2.2, which shows whether these systems have been translated into logic or not. Because most of these systems were tested on the RTE, the figures in the last two columns show the version of the RTE test and how accurate these systems are in these versions of the test. However, these systems are not comparable, as they were evaluated on different data sets, as shown in the fourth column. An observation that we made on these systems is that they significantly different from each other but tend to share many resources, as well as underlying structures [Padó et al., 2015]. Padó et al.

[2015] provided a specification for an architecture called 'EXCITEMENT'[13], accompanied by an implementation, which provides most of the required components for new RTE systems.

2.5 Summary of the chapter

This chapter begins with defining the terminology of entailment in general and how it works in human brains. Computational linguistic experts have tried to mimic this notion of entailment and develop computer systems that apply entailment. Nonetheless, computer programs have a limited ability in applying a set of rules that are known to be reliable rules that lead to desired goals. This process of proof, however, is very similar to the notion of entailment as it is possible to develop a code that checks whether A is entailed from B or not. The next subsection discusses using a logical-based approach that attempts to translate input sentences into logical formulae, and then applies the proving system using theorem provers through seeking help from some inference rules. It was clear that the step of meaning representation in a logical-based approach remains a difficult problem to solve. Then the proofs in FOL are discussed with a brief consideration of automated reasoning using two well-known methods: resolution and tableau, which are both dependent on the idea of refutation of the hypothesis and trying to get proof. Theorem provers require some kind of inference rules, and there are some resources that contain such rules, for instance, commonsense knowledge bases, such as OMCS, ConceptNet, YAGO, and DIRT. This deep-but-brittle method is considered to be high in precision and low in recall approach.

We also discuss TexTailment techniques that are used either at the surface string level or at the level of dependency trees. These techniques are namely: bag of words, string edit distance, and tree edit distance approaches. These systems are shallow but robust; they are shallow, as they are only applying matching at the surface level and they have high recall but low precision.

In brief, we have two approaches: one of them with great precision but poor recall and the other with poor precision and good recall. A possible solution that people have considered in order to overcome the problems of these approaches is to find an in-between approach. The section about these hybrid solutions is divided into two sections: (i) less-shallow approaches and (ii) more robust, deep approaches. In each section, we describe some systems that have proposed different solutions in that regard, followed by what our system is going to be and how it is related to what we have seen in the literature review. Table 2.3 shows a clear comparison between the similar levels that each of the three approaches follow, i.e., shallow-but-robust, current system and deep-but-brittle. Each of these approaches receives text as input, then there is a step of normalisation, where the shallow and the current systems use syntactic parsers, while deep approach systems employ a semantic parser. Then, after applying some normalisation to the

[13]https://sites.google.com/site/excitementproject/

output of the parsers, the shallow approach applies approximate matching techniques, while the current and the deep systems apply theorem provers. In Chapter 3, there is a discussion about which parser is to be used, followed by Chapters 4 and 5, which concern the inference engine step. Chapter 4 presents the inference engine that is built to do some approximate matching between given sentences, and Chapter 5 demonstrates how we built a database for commonsense knowledge.

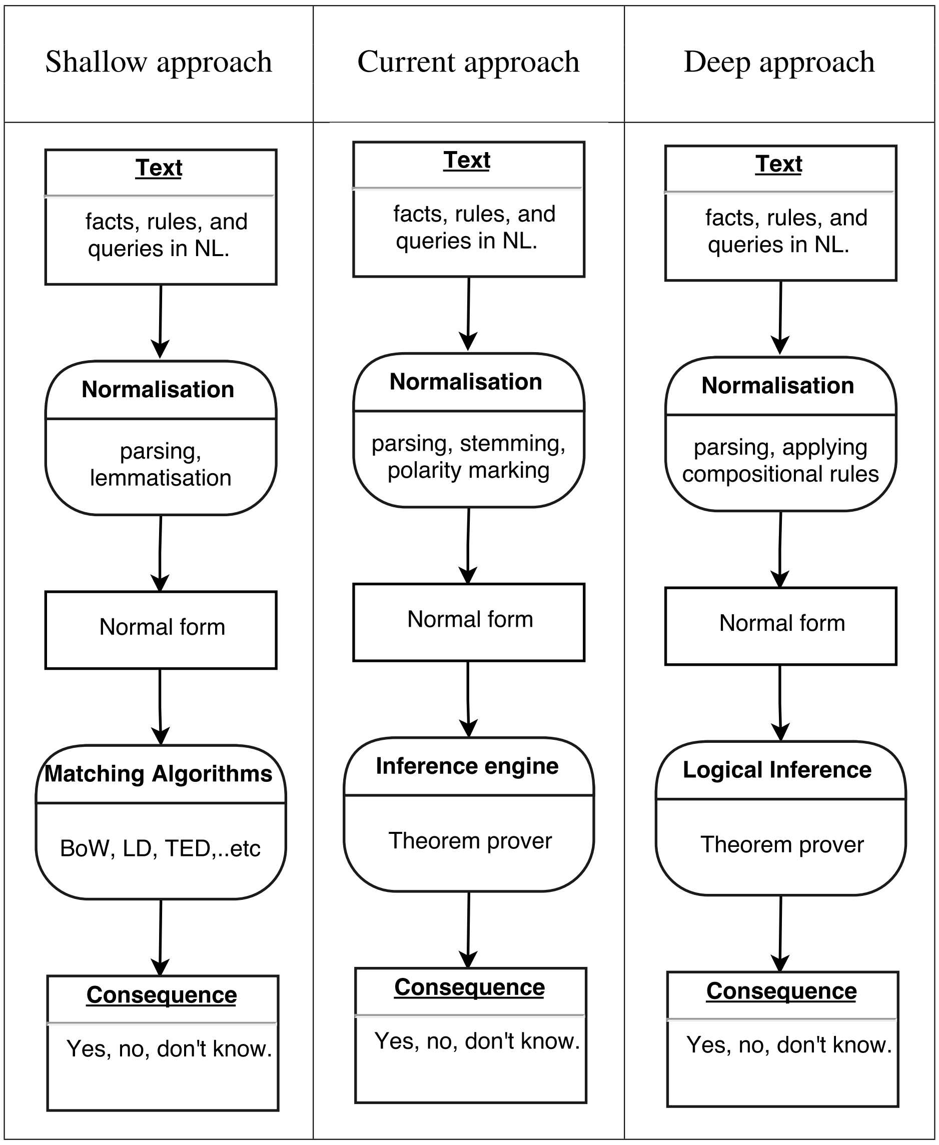

Table 2.3: Architecture for three entailment approaches

Chapter 3

Syntactic parsing

3.1 Overview

This chapter discusses parsing, the process of automatically analysing a given sentence to build its data structure, i.e., a parse tree. A fundamental question in linguistics concerns whether language is simply a collection of words; the answer is no, as rearranging the words contained in a sentence will often create a nonsense statement. Therefore, grammatical rules exist to describe the possible arrangements of words and their grouping. For instance, a sentence must include a subject and a predicate; the subject can typically be a noun phrase (e.g., *'she', 'the cat', 'John'*), while the predicate is a verb phrase (e.g., *'arrived', 'went away', 'had dinner'*). One can combine these noun phrases with any verb phrases to form a syntactically correct sentence (e.g., *'the cat went away' or 'John had dinner'*). Thus, if learning a new word, a learner will be asked to feature that word in a sentence and would first be asked about the type or class that can define the word. For example, when taking the word 'wug'[1] as a noun, one can say *'I can see two wugs'*. Similarly, if testing this sentence on an off-the-shelf parser (e.g., Stanford parser[2]), the output can be presented as seen in Figure 3.1.

[1]This word is taken from 'The WUG test' [Berko, 1958].

[2]http://nlp.stanford.edu:8080/parser/index.jsp

```
(ROOT

    (S

      (NP (PRP I))

      (VP (MD can)

        (VP (VB see)

          (NP (CD two) (NNS wugs))))))
```

Figure 3.1: Output parsing tree from a phrase-structure parser (Stanford parser)

The above shape illustrates a parse tree that contains a part-of-speech tag associated with each word, as in (PRP I), which defines *'I'* as a personal pronoun and (MD can) means that *'can'* is a modal verb, and so on. Despite the word *'wugs'* not being an existing English word, it can be tagged as a noun in this parser, as unknown English words tend to be proper nouns [Tseng et al., 2005], especially if the position of that unknown word in the given context is more likely to be occupied by a noun [Bird et al., 2009]. In addition, some characteristics help to determine certain word classes, typically sought by these taggers when assigning a part of speech. Looking at previous tag examples may help to decide the appropriate tag for the current one; for example, words following determiners are usually nouns (e.g., *'the man'*), words following 'cannot' are mostly verbs (e.g., *'cannot sleep'*) and words featured between a determiner and a noun are often adjectives (e.g., *'the old man'*). However, as seen in Figure 3.1, certain acronyms exist, such as (NP and VP), which represent the *constituents* of these words or phrases mentioned in phrase-structure framework in Section 3.2.1. However, there are two widely-used frameworks for processing natural language sentences, which shall be discussed in the following section.

3.2 Grammatical frameworks

Figures 3.2 and 3.3 contain examples of so-called 'parse trees'. Generally, parse trees are constructed depending on specific rules, and in this section, a brief discussion shall be presented about two well-known frameworks: (i) phrase-structure framework (PSF), of which an example is given in Figure 3.2, and (ii) dependency framework (DF), which shall be used as one of the applications in our system (see Figure 3.6).

3.2.1 Phrase-structure framework

Phrase-structure grammar—also known as constituency grammar—is a form of parsing natural language sentences depending on their constituency relation. This basically works by mapping between an input natural language sentence and one or more phrases; it models a sentence as a tree in which words are grouped into phrases (constituents) that are then classified by structural categories, such as NP, VP etc. Section 3.2.1.1 details some tests used to decide whether a sequence of words denotes a constituent or not, followed by Section 3.2.1.2, which indicates the notion of grammars.

3.2.1.1 Constituency

A constituent, as defined by Ceusters et al. [1999], is a word or group of words used in a statement. As per the previous example in Figure 3.1, we have two noun phrases: one is a single word `(NP (PRP I))` and the other is a group of words `(NP (CD two) (NNS wugs))`. In English, there are various forms to construct a noun phrase, such as: nouns (e.g., *'**people** like fast cars'*) or pronouns (e.g., *'**it** is getting late'*). Other examples of noun phrases that contain more than one word may come as a determiner with a noun (e.g., *'**the man** is here'*), with an adjective (e.g., *'**the best team** won'*) or beginning with a quantifier (e.g., *'**all my family** are supporting me'*). As noun phrases can also be more complicated (e.g., *'I ate **a loaf of nice fresh brown bread**'*), it is essential to know the limits for a noun phrase. Therefore, some constituency tests have been developed that can guide one to the right limits, some of which are mentioned in Phillips [2003], Carnie [2013] Allerton [1979], Borsley [2014], Napoli [1993], Burton-Roberts et al. [2016], Radford [2004], and Haegeman [2009] and are as follows:

- Coordination test: This test states that if two words are combined with a conjunction, then they are from the same category, e.g., *'Eyad and Dahoomi are going'*. Here, the two words featured before and after the conjunction 'and' represent a noun phrase; thus, if the tree is drawn for this sentence—using the Stanford parser—it will have the form as seen in Figure 3.2.

- Pronoun test: This test identifies whether it is possible that one word or more can be replaced by a pronoun, such as changing 'Eyad and Dahoomi' in the previous test with the pronoun 'they'; if so, the unit is a constituent (see Figure 3.3). As seen in Figure 3.2 and Figure 3.3, they have similar trees that explain the point of this test.

- Question by repetition test: Suppose someone states the following sentence, *'I have seen the president'*, there are various sequences of words (e.g., 'seen the president', 'seen the', 'the president'), but if people try to emphasise what they have heard, they would ask 'the president?', which is a constituent. On the other hand, 'seen the' does not come across as plausible dialogue; therefore, it is not a constituent.

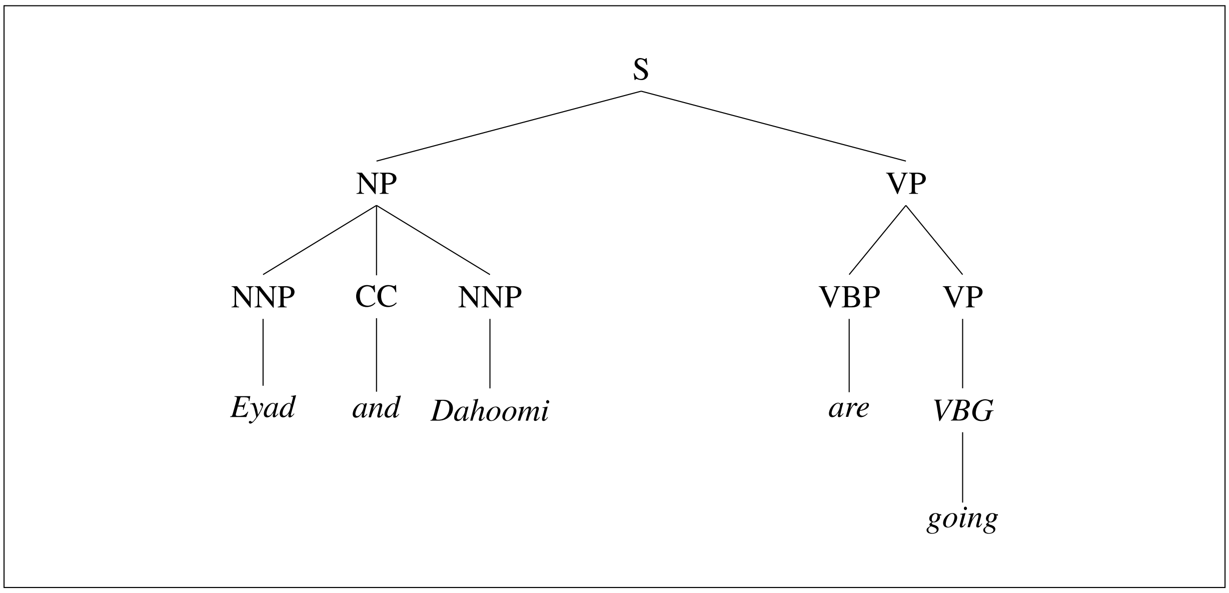

Figure 3.2: Parsing conjunctive constituents

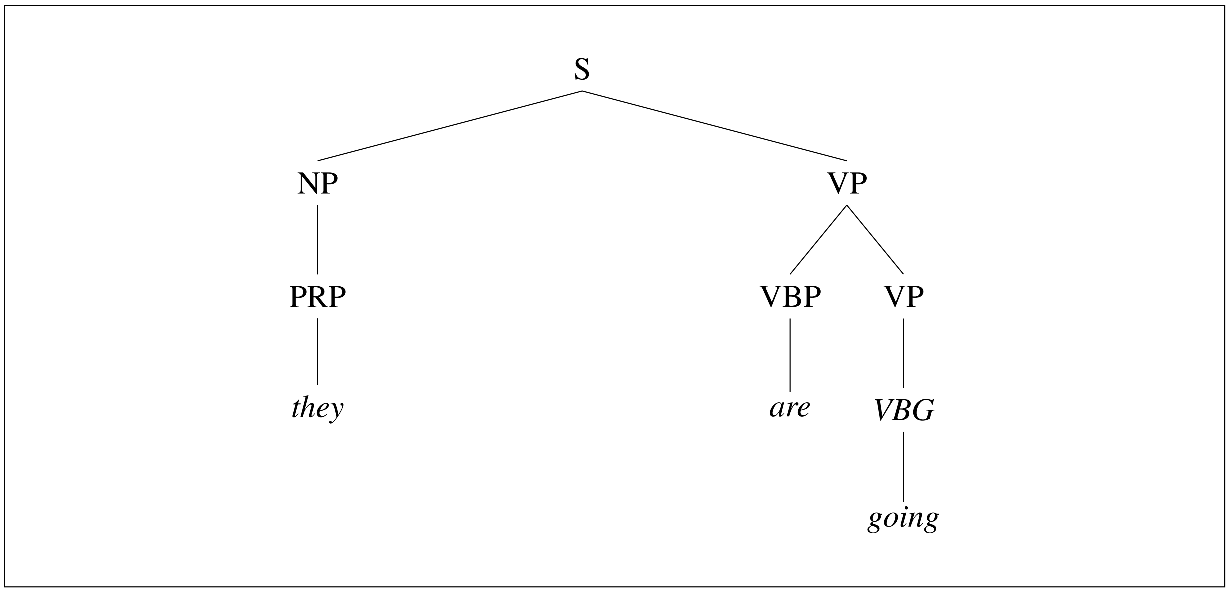

Figure 3.3: Parsing pronoun constituents

- Topicalisation test: This allows one to check if something is a constituent by fronting it at the start of a sentence. That means, instead of stating *'I have seen the president'*, it can be said as *'the president, I have seen'*; therefore, 'the president' is a constituent.
- Question test: If a phrase can be replaced with a questioning word (e.g., *who*), that is also considered to be an indication of the phrase being a constituent (e.g., *'who have I seen?'*); the answer is*'the president'*.

3.2.1.2 Phrase-structure grammars

Regarding this type of grammar, functional relations (e.g., subject and object) can be identified in terms of structural configurations (e.g., 'NP under S', and 'NP under VP') [Gildea and Palmer, 2002].

Suppose that we have the following grammar:

```
S → NP VP PU
NP → JJ NN | JJ NNS
VP → VBD NP PP
PP → IN NP
```

This will assign the structure in Figure 3.4 to the first sentence (a) in the following Example 3.1, featured in Kübler et al. [2009a]:

(3.1) a. *Economic news had little effect on financial markets.*

b. *A hearing is scheduled on the issue today.*

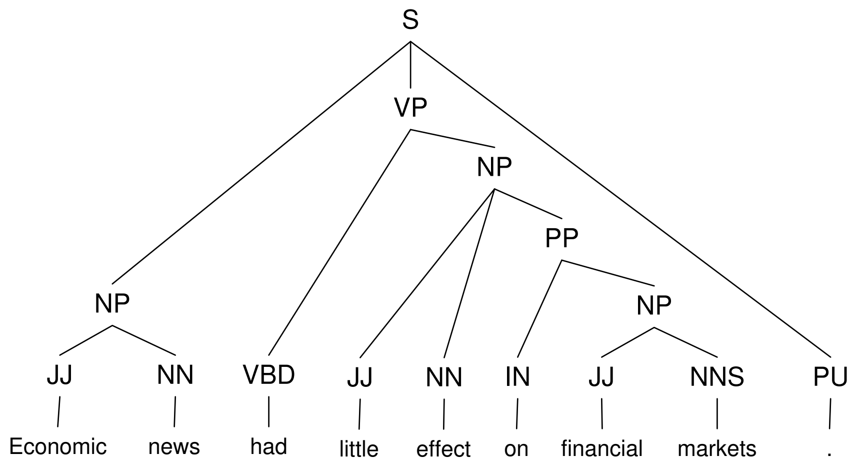

Figure 3.4: Phrase-structure tree for Example 3.1-a [Kübler et al., 2009a]

Regarding the sentence (b) from Example 3.1, when attempting to draw the tree, the phrase *'on the issue'* would be a modifier on *'scheduled'*. There is no technique to write simple phrase-structure grammar that will attach *'on the issue'* to its right parent (i.e., *'hearing'*).

3.2.2 Dependency framework

In recent years, there has been a vast surge of interest in using dependency frameworks for parsing natural language syntax, which has led to developing a few great parsers, such as

MaltParser [Nivre et al., 2006]. The syntactic structure of this framework consists of lexical items that can be linked by binary asymmetric relations called *'dependencies'*. This depends on the idea that in any given sentence, all but one word depends on other words; whereas the word that does not have a parent is labelled the root. Some constraints of a dependency tree exist that have been formulated by Robinson [1970] and these are depicted below:

1. One and only one element is independent.
2. All others depend directly on some element.
3. No element depends directly on more than one other.
4. If A depends directly on B and some element C intervenes between them (in the linear order of the string), then C depends directly on A or B or some other intervening element.

3.2.2.1 Heads and roots

For any produced dependency tree, there is a relation (a dependency relation) between each of its components. For instance, if there is a small sentence containing only two words (e.g., *'financial markets'*), then the parent[3] is the noun *'markets'*, while the adjective *'financial'* acts as its child[4], since it somehow modifies the parent *'markets'*.

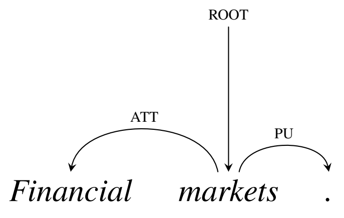

Figure 3.5: A dependency graph for the phrase *'financial market'*

As seen in Figure 3.5, there is only one root that fulfils the first point in Robinson axioms, as there is only one independent element (i.e., *'markets'*).

3.2.2.2 Single-headedness elements

Axioms (2) and (3) can be merged at this point, as they are, somehow, related to one another. As seen in Figure 3.5, all elements—other than the root—are linked to other elements. The punctuation mark (.) depends on the root *'markets'* which is, in the meantime, a parent of the

[3]Also called a head, a governor or a regent.

[4]It is also called a dependent, a modifier or a subordinate

other node *'financial'*. Consequently, axiom (2) is applied in this figure. In addition, there is no element in this tree that is depending on more than one parent, as the punctuation mark (.) depends only on one element; this is the case for the other word *'financial'* too. By reaching this point, all top three axioms are fulfilled within this tree. However, these points are enough to produce well-formed dependency trees.

3.2.2.3 Projectivity and non-projectivity

The fourth point of Robinson axioms is a common constraint on dependency representation of natural language, in which the dependency tree is restricted with direct and non-overlapping arcs between its nodes. An instance of such projectivity can be obtained when we draw a dependency tree, for Example 3.1-a (as seen in Figure 3.6).

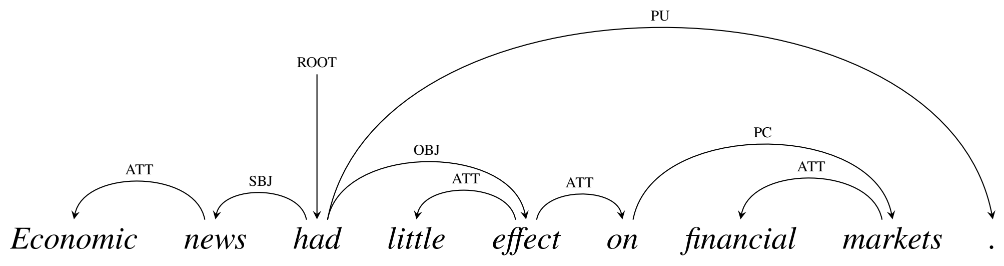

Figure 3.6: Dependency parsing tree for Example 3.1-a [Kübler et al., 2009a]

The arcs in Figure 3.6—between pairs of vertices—represent the dependency relations pointing from the head to the dependent, while labels on the arcs indicate the dependency types. Hence, it is clearly seen in Figure 3.6 that all four constraints are attained, as there is only one root (i.e., '*had*'), with all other nodes being children of some parents; no child has more than one parent, and finally, no crosses exist between the arcs.

Formally, for an input sentence with n words, $S = w_1, ..., w_n$, a legal dependency tree is $T = (V, A)$, consisting of a set V of vertex $V \subseteq \{w_1, ..., w_n\}$ and a set A of dependency relations $A \subseteq V \times R \times V$, R is a set of dependency types $R \subseteq \{r_1, ..., r_{n-1}\}$, if $(w_i, r, w_j) \in A$, then $(wi, r', wj) \notin A$ for all $r \neq r'$. The dependency tree has n vertices and each vertex corresponds to a word in the sentence; i.e., for any $i \leq n(i, r, 0) \notin A$. Since the structure of a dependency tree is *acyclic*, meaning that it does not contain cycles - i.e., for all $w_i, w_j \in A$ if $w_i \to w_j$ - then it is not the case that $wj \to^* w_i$ Kübler et al. [2009b]. Thus, the vertices sets and dependency relations in Figure 3.6 are as follows:

```
V = {Economic, news, had, little, effect, on, financial, markets, .}
A = {(news, ATT, Economic), (had, SBJ, news),
     (effect, ATT, little), (had, OBJ, effect), (effect, ATT, on),
     (markets, ATT, financial), (on, PC, markets), (had, PU, .)}
```

From the set of relations, the verb $'had'$ is the head of two nouns: $'news'$ with the dependency type SBJ and *'effect'* with the OBJ dependency type.
According to Alabbas [2013], a tree is considered projective if, and only if, every dependency arc $(i, r, j) \in A$ and node $x \in V, i < x < j$ or $j < x < i$; then there is a subset of dependency arcs $\{(i, r, i_1), (i_1, r_1, i_2), ..., (i_{x-1}, r_{x-1}, i_x)\} \in A$ such that $i_x = x$.
However, in the case that the fourth constraint is not satisfied (i.e., projectivity), there is an opposite notion of *'non-projectivity'*; therefore, the crosses between dependency arcs occur, as seen in Figure 3.7. This example, as per the analysis of Kübler et al. [2009a], contains an instance of long-distance dependency; as seen in Figure 3.7, there is a prepositional modifier *'on the issue'* on the noun *'hearing'*. Normally, prepositional modifiers on nouns are adjacent to their noun, yet this one has been moved to a position between the verb *'scheduled'* and its modifier *'today'*.

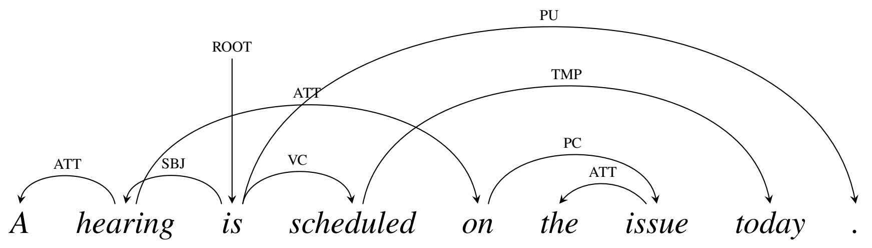

Figure 3.7: Dependency parsing tree for Example 3.1-a [Kübler et al., 2009a]

3.3 Parsing approaches

3.3.1 Rule-based parsing

Rule-based parsing was the classical method of parsing three decades ago, meaning that to parse a well-formed sentence or phrase, one must find a structure that matches handwritten rules. These rules can be transformed into Chomsky's normal form (CNF), as in the CKY[5]

[5]http://martinlaz.github.io/demos/cky.html

parser [Hopcroft et al., 2001], or not, as in early parsing [Earley, 1970]. Mimicking the trace of a bottom-up rule-based algorithm given by Burridge and Stebbins [2015] within the first sentence (a) in Example 3.1, the following can be concluded in Figure 3.8:

```
 1: Take first word:  Economic_JJ.
 2: Check if the word, on its own, can be RHS of any rule:  NO.
 3: Check if the word is the first element of the RHS of any rule:  YES.
 4: Assume NP.
 5: Take the next word:  news_NN.
 6: Check whether the word can be the next element in NP: YES.
 7: Continue to assume NP.
 8: Check whether the rule requires another element:  NO.
 9: Assign label NP to substring 'Economic news'.
10: Check whether NP is the first element of the RHS of any rule:  YES.
11: Look for VP.
12: Take the next word:  had_VBD.
13: Check whether the word can be the only element in VP: NO.
14: Check if the word is the first element of the RHS of VP: YES.
15: Take the next word:  little_JJ.
16: Check whether the word can be the next element in VP: NO.
17: Parse fails, start again.
18: Look for VP.
19: Take the next word:  had_VBD.
20: Check whether the next element on its own can be RHS of any rule:  YES.
21: Check if the word is the first element of the RHS of VP: YES.
22: Take the next word:  little_JJ.
23: Check if the word on its own can be RHS of any rule:  NO.
24: Check if the word is the first element of the RHS of any rule:  YES.
25: Assume NP.
26: Take the next word:  effect_NN.
27: Check whether the word can be the next element in NP: YES.
28: Continue to assume NP.
29: Take the next word:  on_IN.
30: Check whether the word can be the next element in VP: NO.
31: Parse fails, start again.
32: ............
```

Figure 3.8: A trace of bottom-up rule-based parsing steps from Burridge and Stebbins [2015]

Figure 3.8 indicates a trace of a bottom-up rule-based parser that illustrates its potentially bad behaviour. Hence, all other strategies have similar issues, and consequently, rule-based parsing tends to be exceedingly slow.

3.3.2 Data-driven parsing

Recently, instead of depending on grammar parsers, linguists tend to benefit from machine-learning techniques to train their parsers on large corpora containing annotated parsed trees; this is the so-called data-driven approach. One important resource used in building parsers is the Penn Treebank[6] [Marcus et al., 1993]. The Penn Treebank is a large collection of 40,000 training sentences and 2,400 test sentences that are parsed and manually annotated using the Penn Treebank tag set. Most of the sentences used in this treebank are from Wall Street Journal news stories, as well as some spoken conversations; for example, this is one sentence used in the Penn Treebank (Section 12, sentence 11) from the Wall Street Journal:

> *"Because the CD had an effective yield of 13.4% when it was issued in 1984, and interest rates in general had declined sharply since then, part of the price Dr. Blumenfeld paid was a premium - an additional amount on top of the CD's base value plus accrued interest that represented the CD's increased market value".*

Figure 3.9 shows the beginning of the parsing tree for this example from the Penn Treebank.

Regarding these trees in Figure 3.9, the parsers have been trained and measured for their accuracy measures, an instance of which is MaltParser[7] [Nivre et al., 2006] and MSTParser[8].

3.4 Why data-driven dependency parsing?

For our system, we employ dependency parsing because the trees' output by dependency parsers encodes the predicate-argument structure in a transparent manner, which supports direct comparison of the meanings of the input texts [Alabbas, 2013].

Given that the aim of this work is to carry out inference over arbitrary texts, we have chosen to use a data-driven algorithm for obtaining such trees. As stated by Kübler et al. [2009b], most data-driven approaches assume that any input sentence is a valid sentence and it is the parser's duty to find the most plausible dependency structure for the input, regardless of how unlikely the result is. Taking such an approach means that our inference engine will always be given

[6]http://www.cis.upenn.edu/ treebank/home.html
[7]Freely available at: http://www.maltparser.org
[8]Freely available at: http://www.seas.upenn.edu/ strctlrn/MSTParser/MSTParser.html

```
(IN Because)
(S
  (S
    (NP-SBJ (DT the) (NNP CD))
    (VP
      (VBD had)
      (NP
        (NP (DT an) (JJ effective) (NN yield))
        (PP (IN of) (NP (CD 13.4) (NN %))))
      (SBAR-TMP
        (WHADVP-4 (WRB when))
        (S
          (NP-SBJ-1 (PRP it))
          (VP
            (VBD was)
            (VP
              (VBN issued)
              (NP (-NONE- *-1))
              (PP-TMP (IN in) (NP (CD 1984)))
              (ADVP-TMP (-NONE- *T*-4)))))))))
                ......
```

Figure 3.9: Example from the Penn Treebank (12:11)

something to work, which would not generally be the case if we used a rule-based parser.

The theorem prover itself is written in Python. It would therefore be helpful if the parser were also written in Python, since this would allow smooth integration of the two. This led to the decision to use a Python reimplementation of the basic algorithm underlying MaltParser.

MaltParser

MaltParser[9] is a system for data-driven dependency parsing that can be used to induce a parsing model from treebank data and to parse new data using an induced model. This parser is language independent and has been implemented for certain languages, such as Swedish, English and Arabic. As stated by Nivre et al. [2006], the methodology for this parser is dependent on three key themes, as depicted below:

[9]The acronym describes the ‘Models and Algorithms for Language Technology Parser’; it is freely available at: http://www.maltparser.org/.

1. Deterministic parsing algorithms for building dependency graphs [Yamada and Matsumoto, 2003]; [Nivre, 2003].
2. History-based feature models for predicting the next parser action [Black et al., 1992]; [Magerman, 1995]; [Ratnaparkhi, 1997]; [Collins, 2003].
3. Discriminative machine learning to map histories to parser actions [Yamada and Matsumoto, 2003]; [Nivre et al., 2008].

Therefore, this parser consists of a transition system to derive dependency trees through the deterministic parsing algorithms of the above first theme. Then, there is a classifier (the second theme) that deterministically predicts the next transition, given a feature representation of the current parser configuration [Nivre, 2008]. As Alabbas [2013] states, the third theme can be used to train the classifier through a given a set of transition sequences derived from a treebank. This transition-based parser employs the strategy of shift-reduce parsing, in which there are two key data structures (i.e., *STACK* and *QUEUE*) and two types of actions (i.e., *Shift* and *Reduce*). The *STACK* is empty when initialised, while the *QUEUE* is initialised to contain each word from the input sentence. The *Shift* action pushes the top word from the *QUEUE* into the *STACK*, where the *Reduce* action inserts two items from the *STACK* or the *QUEUE*, depending on the direction of the arc relation. As there are two directions, i.e., $LEFT\text{-}ARC_{(r)}$ and $RIGHT\text{-}ARC_{(r)}$, if the relation (r) is starting from the top of the *QUEUE*'s items to the top of the *STACK*'s items, then it is $LEFT\text{-}ARC_{(r)}$ and placed in the *STACK*. On the contrary, if the relation (r) starts from the top of the *STACK*'s items to the top of the *QUEUE*'s items, then it is $RIGHT\text{-}ARC_{(r)}$ and put in the *QUEUE*. The following pseudo-code for MaltParser in Figure 3.10 is taken from Jaf and Ramsay [2015].

```
1. Given an oracle and a string of input text, set the queue
   to the input text and the stack to be empty.
2. Until the queue is empty and the stack has exactly one entry,
   ask the oracle for an action.
   2.a). If the oracle suggests LEFT-ARC(r) then make the head of
         the queue a parent of the  topmost item on the stack and
         pop the stack.
   2.b). If the oracle suggests RIGHT-ARC(r) then make the topmost
         item on the stack the parent of the head of the queue and
         pop the queue.
   2.c). If the oracle suggests a SHIFT then move the head of the
         queue to the top of the stack.
```

Figure 3.10: Pseudo-code for MaltParser

3.5 Summary of the chapter

This chapter has detailed the approaches used when parsing. Firstly, the notion of consistency in natural language sentences has been covered by presenting examples of tests used to gain the right limits of the constituents. The listed tests are: the coordination test, pronoun test, question by repetition test, topicalisation test and question test. Then, two well-known grammatical frameworks were discussed: (i) the phrase-structure grammar that was addressed as a parsing method that depends on the constituents of input sentences, and (ii) the dependency grammar that captures the dependencies between individual words, followed by a discussion of parsing approaches: rule-based parsing and data-driven parsing. Rule-based parsing depends on grammar that attempts to match the input sentence with that grammar (from top to bottom or vice versa). For the top-down approach, it starts from ‘S’ (the root for the parsed tree) to reach the word level, while the bottom-up way begins from the word level to reach the root. The data-driven approach represents another approach that trains the parsers on a big corpus (e.g., the Penn Treebank) which then parses the input text, depending on the most likely matching tree from the training set. Finally, there is a brief discussion to answer why we prefer to use the data-driven dependency parsing. As we need a resulting tree for any input text, we have to avoid parsers that depend on rules. In addition, the dependency-based approach is considered fast and robust, as well as its representation being considered effective in the textual entailment area, so we have chosen a dependency parser (MaltParser). After parsing sentences, we must test the inferential validity, and in this system, we aim to apply the theorem provers to the resultant parsed trees. Theorem provers will be discussed more extensively in the following chapter.

Chapter 4

Theorem proving over dependency trees

4.1 Overview

Theorem provers are the inference engines used by logical entailment applications to identify the inferential validity of any given relation. These relations are usually written in some kind of logic, for instance, FOL. Consequently, by employing some proof theories (e.g., *'Tableaux'* and *'Resolution'* in Section 2.2.2), these theorem provers return results of either 'True', if there is an entailment hold or 'False', if otherwise.
Since we are using dependency trees as the input to our inference engine, we cannot use any of the existing unchanged theorem provers. Thus, two issues are encountered. The first problem with shallow approaches is that one cannot use more than one step, thus preventing us benefiting from background knowledge. On the other hand, the issue with deep approaches is that there is no grammar set is complete. Nonetheless, the two approaches have completely different problems, which allow us to devise a solution between these two methods. To address this problem, one of two broad actions can be taken: either to make a shallow inference less superficial or to make a deep inference more robust. The latter solution has not been considered, as the main problem for the deep approach is being brittle; therefore, we decided to use the former solution by making the shallow approach less superficial. The idea here is to build a theorem prover that can deal with dependency parser outputs.

4.2 Building a theorem prover

To make an inference system that can benefit from background knowledge, we must utilise a similar idea to existing theorem provers. As the background knowledge being used is extracted from English dictionaries (see Chapter 5), they are typically Horn clauses. Therefore, we will attempt to mimic the strategy that works well for the Horn subset of FOL backward chaining. Backward chaining suits theorem provers, as they have a goal that needs to be proved [Russell and Norvig, 2016].

The reason why backward chaining for Horn clause reasoning is successful is trying to match the current goal with the head of another rule (see Equation 4.1). However, the step of matching may vary according to the representational language.

$$\begin{gathered}\texttt{prove(A)} \therefore \texttt{B} \rightarrow A' \texttt{, A (match) A',} \\ \texttt{prove (B)}\end{gathered} \tag{4.1}$$

For systems that use propositional logic—as the representational language—their matching for a Horn clause refers to (identity) between A and A', as in Equation 4.2.

$$\begin{gathered}\texttt{prove(A)} \therefore \texttt{B} \rightarrow A' \texttt{, A = A',} \\ \texttt{prove (B)}\end{gathered} \tag{4.2}$$

While in FOL representations, the matching between A and A' is a (unification) between the two sentences (see Equation 4.2).

$$\begin{gathered}\texttt{prove(A)} \therefore \texttt{B} \rightarrow A' \texttt{, A} \oplus \texttt{A',} \\ \texttt{prove (B)}\end{gathered} \tag{4.3}$$

Previous Equations 4.2 and 4.3 indicate the matching relation in Equation 4.1, and that can be customised to different forms of matching relations. Since we use dependency trees as inputs, we utilise a notion of unification as an asymmetric relationship involving various kinds of relaxing and the notion of matching; this includes using WordNet hypernyms, skipping adjuncts as adjective modifiers and skipping prepositional modifiers. Therefore, the matching relation that we are going to use in our backward chaining is an approximate matching (≈) between the head and a sub-goal; formally, consider Equation 4.4.

$$\begin{gathered}\texttt{prove(A)} \therefore \texttt{B} \rightarrow \texttt{A' , A} \approx \texttt{A',} \\ \texttt{prove (B)}\end{gathered} \tag{4.4}$$

Another method of proving a hypothesis is if it has a direct relationship with one of the facts, i.e., the fact A' is identified with the hypothesis A in prepositional logic (see Equation 4.5).

$$\texttt{prove(A)} \therefore \texttt{fact (A'), A = A'} \tag{4.5}$$

Similarly, if there is a unification (entailment) between the two literals A and A' in the FOL (see Equation 4.6), and therefore, in our matching algorithm, if there is an approximate matching holds between them (see Equation 4.7).

$$\texttt{prove(A)} \therefore \texttt{fact (A'), A} \oplus \texttt{A'} \tag{4.6}$$

$$\texttt{prove(A)} \therefore \texttt{fact (A'), A} \approx \texttt{A'} \tag{4.7}$$

Systems using backward chaining are, of course, non-deterministic, as the first rule these algorithms try does not always lead to a solution. However, they will not know whether it is a useful rule until they try it out. If unsuccessful, they must backtrack to source a different rule to attempt.

The approximate matching that we are using also turns out to be non-deterministic, therefore we must find a way to seamlessly manage the non-determinism of the backward chaining algorithm, together with the approximate matching algorithm. The way to handle this is by using continuation programming, in which we add an extra 'continuation' argument, and the value of this algorithm is passed on to its extra continuation argument. In incidences when there are three steps of inference (three rules) between the *'goal'* and the *'text'*, and the system manages to unify the goal with a head of one rule, we can get to the continuation argument to use the current value as the new goal. Suppose we have a text *A*, a hypothesis *B* and a set of rules that lead to a unification in three steps, as seen in Figure 4.1.

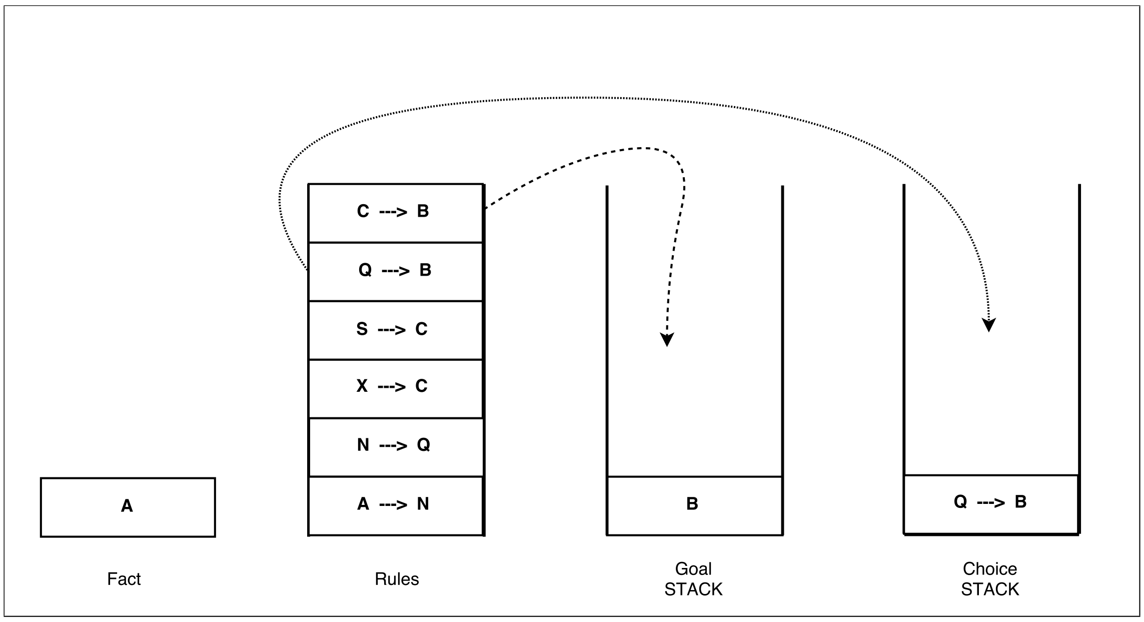

Figure 4.1: World-knowledge representation example

When the system starts to backtrack, a 'goal stack' is generated to contain the current objective, which is a precursor of a rule that its head is proved to be unified with the current goal. When it finds that there is a rule ($C \rightarrow B$), we try to match B with B. In case this matching succeeds, then the current goal is the antecedent of the proved rule, which is C in this case (See Figure 4.2). Another stack is then introduced, called the 'choice stack', which is used to tell all about the next rule we should use after the current action.

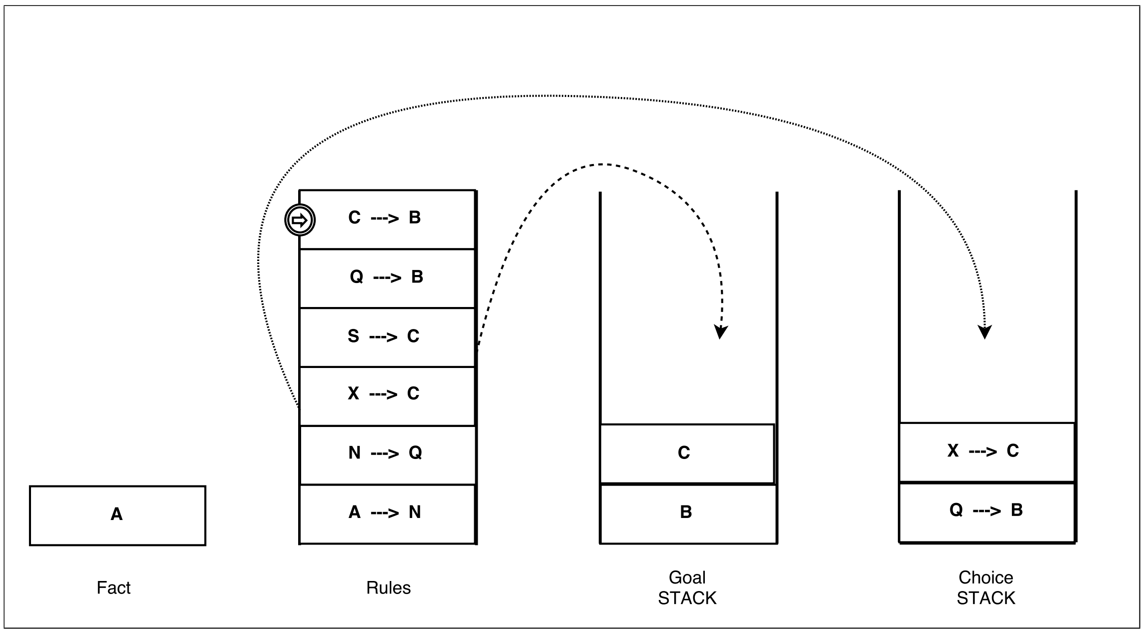

Figure 4.2: Continuing on the backtracking algorithm - 1

Following this step, the system will go back to the rules and begin from the choice available to it, before attempting the first suitable rule ($\mathsf{S} \rightarrow \mathsf{C}$) and pushing the following one into the choice stack (as seen in Figure 4.2). Consequently, both rules will have C as their head will fail, shoving off the last item in the goal stack, which is C. At this point, the current goal is the original goal B (as seen in Figure 4.3).

This time, the system will find the fact A by following the map $Q \rightarrow B \Rightarrow N \rightarrow Q \Rightarrow A \rightarrow N$. Upon successfully unifying all nodes and pushing them back into the stack, the next step in Figure 4.2 is taken and so on.

For our system, the ordinary normal calling stack can be used as our goal stack. However, it is hard to maintain the choice stack effectively, as pushing things off the call stack does not necessarily insert stack choice. As an alternative, we tend to use the programming language calling stack as the choice stack. Management of the goal stack is achieved by building a 'continuation', which is a description of what remains to be done; this manages the choice stack efficiently and smoothly for our system. The description of what remains to be done (*WRTBD*) can be captured by building a function out of our current view of what is left to be completed. Suppose *WRTBD* is a description of our current view of what remains to be done and *NTTD*

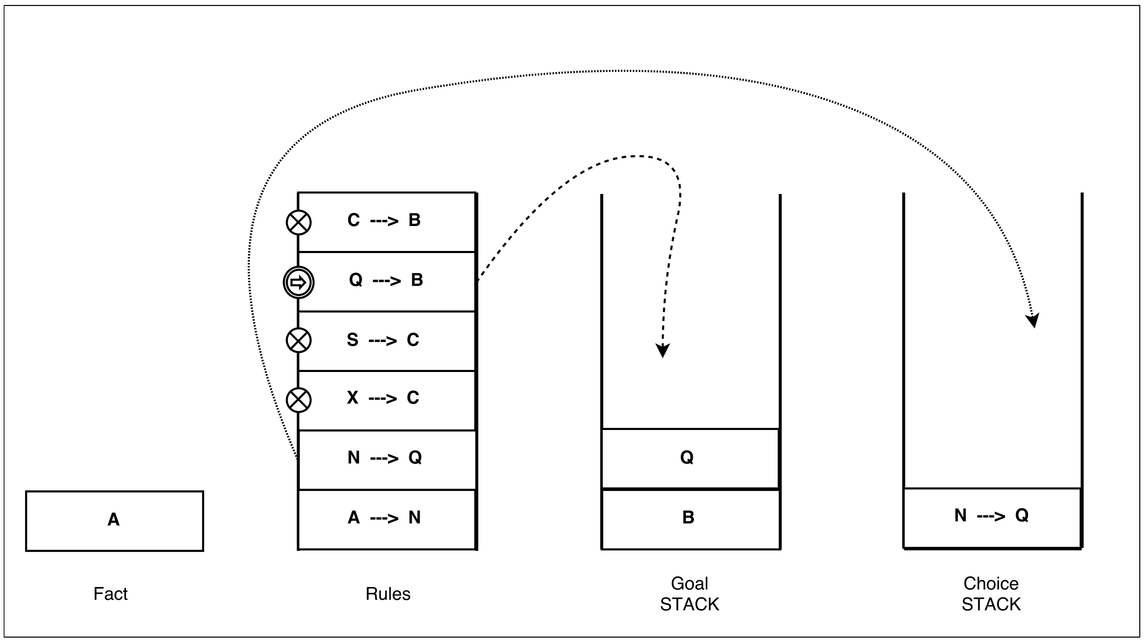

Figure 4.3: Continuing on the backtracking algorithm - 2

is a specification of the next thing to do, `lambda: NTTD (WRTBD)` then states that what our system wants to do is NTTD followed by WRTBD.
Therefore, building 'continuations' of this kind allows our system to suitably manage the goal stack. This all involves taking a slightly non-standard view of what a program does; particularly, a function call returns if it has failed, not if it has succeeded. This allows us to explore a series of choices by specifying them as a sequence of function calls:

```
RULE 1 (contn)
RULE 2 (contn)
RULE 3 (contn)
      .
      .
      .
RULE n (contn)
```

This will explore whether `RULE 1` will lead to a successful outcome, including all tasks currently on the goal stack. We will only return to `RULE 2` if `RULE 1` does not lead to a solution. It is worth noting that we may try a variety of attempts when following up `RULE 1` - all choices that might arise as we look at methods of doing `RULE 1`. Thus, this all means we cannot mark (success) by simply returning up the standard call stack. We must initialise the 'contn' as a special type of event. If a successful outcome is reached, we report that such an event has occurred and we catch such reports at the outer level of the search.

The other unification algorithm is a non-deterministic algorithm, should the length of the two subtrees that require matching not be equal. In Section 4.2.1.2, we will explore how our algorithm can deal with different length trees, should one tree contain adjuncts, such as adjective modifiers. If we must match two trees, T_1 and T_2, where T_1 is longer than T_2, then there is a possibility for them to match, should the system use a skip-over feature. If the system tries to carry out a match and fails, there is then the possibility to take an action of skipping over where the continuation used is in the matching algorithm. Other than different tree lengths, the matching algorithm is deterministic, as it has some check points; if one is applied for any two given subtrees, then a contn is applied. When comparing two words w_1 and w_2, then the ‘contn’ is applied if these words are either like one another in a similar context ($w_1 = w_2$) or if w_1 is a hypernym of w_2; i.e., $w_1 \sqsubset w_2$. More discussion shall be had about this in Section 4.2.1.1. Variables are used in this system and their values can be words or subtrees; a word in our system is a class that has a form, a tag and a polarity marking, while subtrees usually contain one or more words, and in some cases, they contain variables. The polarity marking is used to identify whether the word came in a positive or a negative context. More discussion about this shall be had in Section 4.2.1.4. When seeking hypernym relations, we rely on WordNet hypernymy relations that cover a huge number of open-class words. However, when trying to enlarge the hypernyms, we cover generalised quantifiers; e.g., *all*$\sqsubset$ *some* (see Section 4.2.1.3) for this topic.

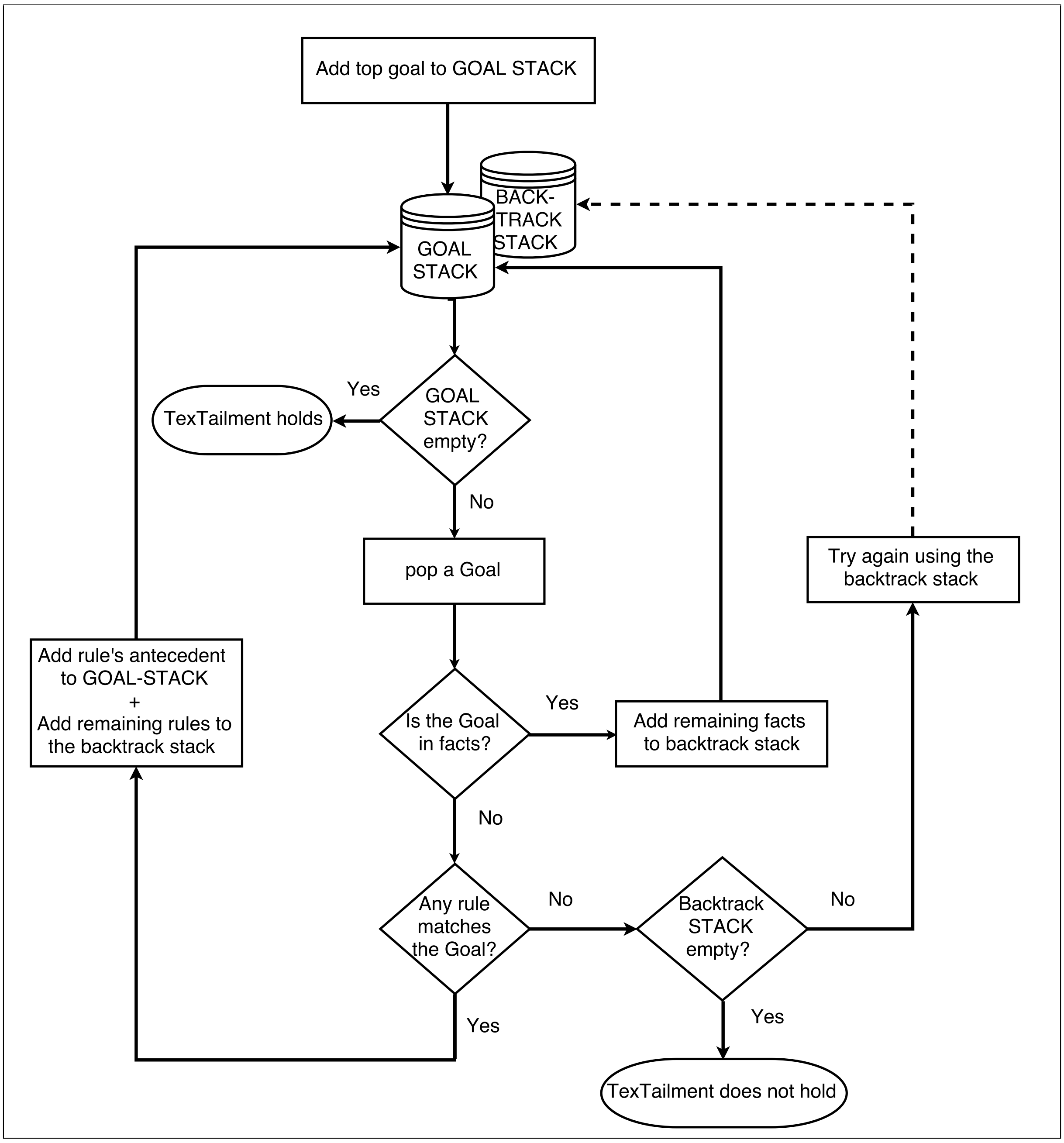
Add top goal to GOAL STACK
BACK-TRACK STACK
GOAL STACK
Yes
TexTailment holds
GOAL STACK empty?
No
pop a Goal
Try again using the backtrack stack
Add rule's antecedent to GOAL-STACK + Add remaining rules to the backtrack stack
Is the Goal in facts?
Yes
Add remaining facts to backtrack stack
No
Any rule matches the Goal?
No
Backtrack STACK empty?
No
Yes
Yes
TexTailment does not hold

Figure 4.4: Theorem prover architecture

Algorithm 1 Pseudo-code for the theorem prover

facts	one or more parsed sentences with marking polarity.
goal	a parsed sentence with marking polarity that we need to prove.
rules	a list of rules that used to get from the to a unification of a fact. The form of rules is $[LHS] \Rightarrow RHS$
hd(goal)	the goal head = [:1]
tl(goal)	the goal tail = [1:]

```
prove ( goal, facts, rules, contn )
    if goal = [] :
        contn()
    if isList(goal) :
        prove(hd(goal), facts, rules, lambda: prove(tl(goal), facts, rules, contn))
    else:
        for f in facts:
            match(goal, f, contn)
            ⊙
        for LHS → RHS in rules:
            match(goal, RHS, lambda: prove(LHS, facts, rules, contn))
            ⊙
```

In Algorithm 1, we reach the points marked with ⊙ only if the previous step fails, i.e., if using that fact or rule did not lead to proof. At that point, we automatically move to the next choice. In Section 4.2.1, we will explore the approximate matching algorithm (≈) that matches trees under certain circumstances, which has already been mentioned in both Figure 4.4 and Algorithm 1.

4.2.1 Approximate matching

The approximately matching relation (≈) has certain rules to follow to derive a match; there are some handwritten rules that the system can take, such as: using WordNet hypernym relations, skipping adjectives and deleting adjuncts. This part is similar to a respectful work carried out by Baroni et al. [2012], who proposed two methods to detect a TexTailment. The first method is to show that there is TexTailment between adjective-noun AN constructions and their head noun N, e.g., *'big car'* ≈ *'car'*. The second path is to find TexTailment on phrases that contain quantifiers Q, e.g., *'many cars'* ≈ *'some cars'*. They consider the TexTailment holds when the noun N in both sides is the same, e.g., *'cars'*, and there is a relation between the two quantifiers, i.e., $Q_1N \approx Q_2N$. For this, we have relaxed some of the constraints mentioned above, as seen in the following lines.

4.2.1.1 Word level matching

One feature that approximate matching applies to this system is accepting relations between two terms A and B; if A $\sqsubset$ B, e.g., (*animal* $\sqsubset$ *cat*). Various TexTailing algorithms have used WordNet hypernym relations, such as Baroni et al. [2012]. However, the relation between A and B here is asymmetric, as A $\sqsubset$ B, but this is not true in the opposite direction. To make use of this feature within our system, we imported the WordNet hypernym relations from the Natural Language Tool Kit (NLTK)[1]. For any two given senses (s_1 and s_2), s_1 cannot be a hypernym or a hyponym of s_2 if it has a different tag. Therefore, for each category of the four open-class words used in the WordNet hypernym relations, we have made a separate folder that contains its relations; consequently, we are left with the following files: *HypN* (for nouns), *HypJ* (for adjectives), *HypR* (for adverbs) and *HypV* (for verbs). This step was taken to reduce the cost to a system when it tries to find a relation for any two given senses. Figure 4.5 illustrates an example of using WordNet hypernym relations between two dependency trees to find a match between the terms *'animal'* and *'cat'*. The sentences used in Figure 4.5 are as follows:

(1) T: *I saw a small animal in the park*
H: *I saw a small cat in the park*

This point was illustrated in the work done by Bowman et al. [2015], when they said it is definitely true to infer *'animal'* from *'dog'*, while it is risky, i.e., not always true, to infer *'puppy'* from *'dog'*. So, the function here only considers if a term T_1 subsumes another term T_2, i.e., $T_1 \sqsubset T_2$, regardless of how similar these two terms are. Thus, we have neither accredited the path length between the terms nor other similarity measurements, such as those developed by Resnik [1995], Lin [1998], Schlicker et al. [2010] and Li et al. [2010]. Nonetheless, the problem of measuring the similarity of this function will be part of any future studies. Note that the unification between hypernym words here is asymmetric, therefore this system will allow a match between (`animal` $\approx$ `cat`), but not (`cat` $\not\approx$ `animal`); this is because if someone saw a small animal, that does not mean that they saw a cat of any size, as it might be a squirrel, a poodle or any other small animal.

[1]Besides WordNet hypernyms, we use WordNet morphy to check the words' roots.

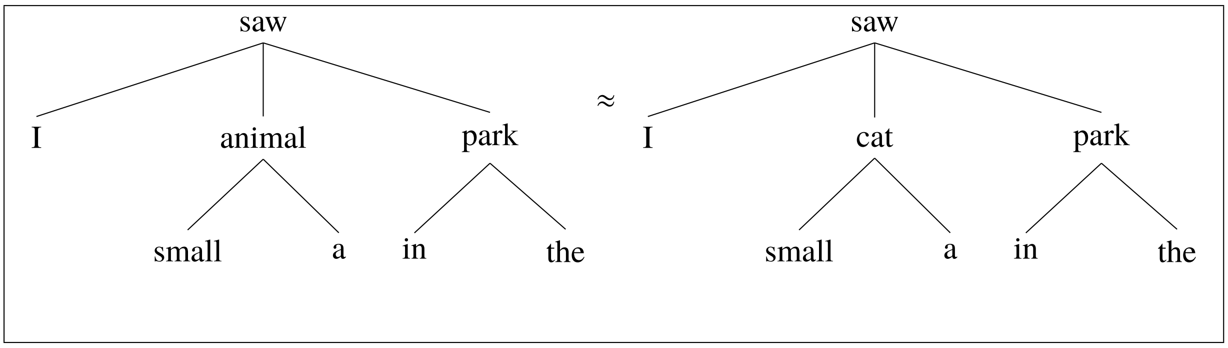

Figure 4.5: An example for hypernym approximate matching

In Figure 4.5, a dependency tree is presented in a hierarchical style, which is common to use with phrasal structure framework. The difference with this tree is that the heads are words, not the constituent symbols, as in Figure 3.4, that are given for phrase-structure trees. So, instead of drawing the dependency trees (as in Figure 4.6), we present them as hierarchy trees, where the top is the root and its dependents are its leaves. We took a similar action to Magnini et al. [2014a], when they generated small dependency trees from bigger ones, as they rely on the relations that are given between nodes in given trees. In their work, they delete the modifiers and reconstruct a tree from the remaining nodes, using the given relations, while in our case, we use these relations as guidelines for the node locations in the resulting hierarchical tree. Representing dependency trees in a hierarchical style without presenting the labels (as in Figure 4.5), makes the observation and the comparison between the two trees easier for the reader.

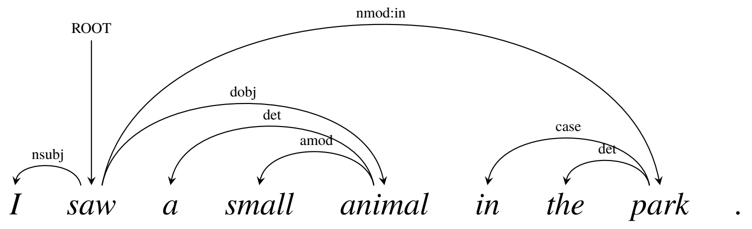

Figure 4.6: Dependency tree for the sentence *'I saw a small animal in the park'*

4.2.1.2 Phrase level matching

In certain English sentences, some additional information (adjuncts) exists that does not affect the meaning of the sentence if omitted. In the following lines, we will discuss the two moves carried out in this regard, which are: skipping adjective modifiers and dropping prepositional phrases.

Skipping adjective modifiers

Inspired by Baroni et al. [2012], another issue that approximate matching handles is skipping some components of noun phrases, such as attributive adjectives. An adjective in English mainly takes one of two cases: it is either used to describe nouns, e.g., *'a small cat'* or is used after a link verb, e.g., *'His car is fast'*. In the former case, it can skip the adjective, as if someone sees a small cat, that means they saw a cat. The opposite, however, is not allowed, as if a man sees a cat, that does not mean that he saw a small cat. To clarify this point, consider the trees in Figure 4.7 for Example 2.

(2) T: *I saw a small cat in the park*
H: *I saw a cat in the park*

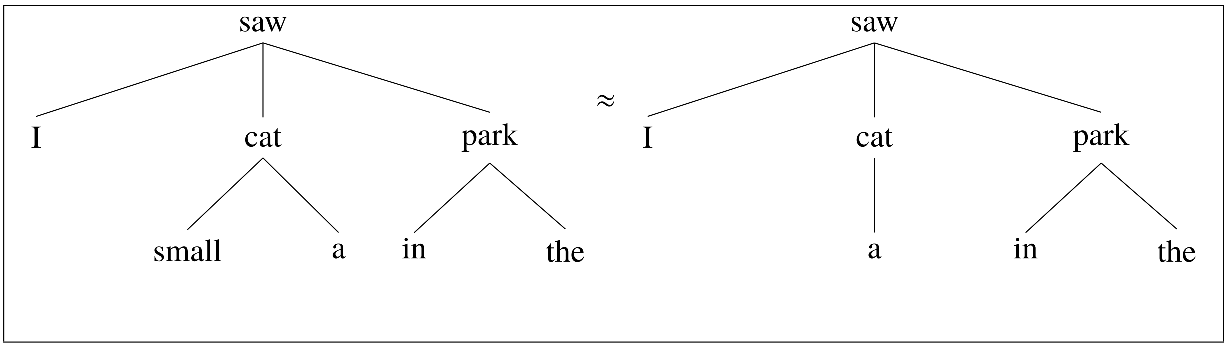

Figure 4.7: An example for skipping adjectives in approximate matching

Therefore, when attempting to unify two subtrees (SB_1, SB_2), in which one contains a noun N_1 and the second contains an adjective followed by a noun JN_2 and $N1 \approx N2$, then we are permitted to skip it. This feature is also asymmetric, as we can do $JN \approx N$, but not $N \approx JN$.

Dropping prepositional phrase modifiers

One more asymmetric feature that can be added to phrasal level matching, that is not covered in Baroni et al. [2012], is dropping prepositional modifiers. These prepositional phrases can be added to English sentences in different forms, for instance (see examples 3, 4, and 5), if someone hears these examples, they would infer that if these examples are true, then the non-bold parts of these examples are also true.

(3) *He met his friend* ***in the gymnasium***.

(4) *I sold my car* ***for £2000***.

(5) *They have arrived* ***on time***.

These prepositional phrases can be detected when we check whether the preposition tag (`IN`) exists in that subtree. Thus, if we have an instance (as in Example 6), all subtrees are unified until we reach a point where SB_2 is an empty list and SB_1 still contains a prepositional phrase that is known by its `IN` tag. The action that we are permitted to take is to drop this prepositional phrase and make a continuation.

(6) P: *I saw a small animal in the park.*
H: *I saw a small animal.*

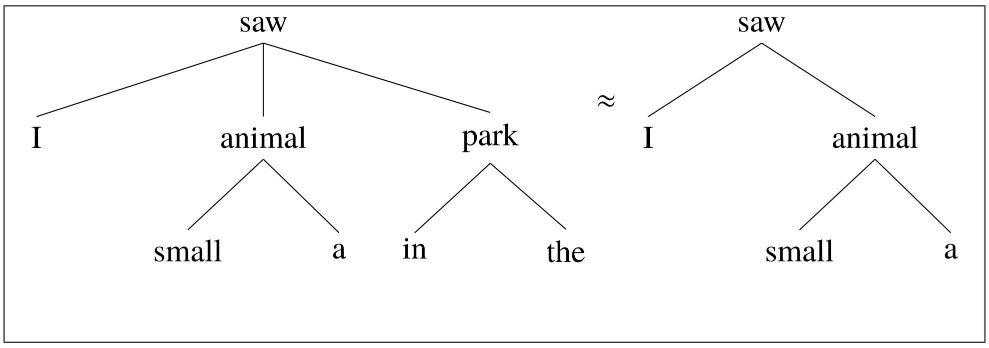

Figure 4.8: An example for skipping adjectives in approximate matching

This feature, however, would fail in some contexts. For example, *'He bought a luxury villa in his dreams'* does not mean *'He bought a luxury villa'*. Such contexts are rare—compared to other cases—so we decided to take this action.
Payne [2006] mentions many other forms that adjuncts may take which can take part in the development of the present work. For instance, an adjunct can exist as a single word (see Example 7) or as an entire clause (see Example 8):

(7) *It was raining **yesterday**.*

(8) *I will call him **after I have had lunch**.*

4.2.1.3 Generalised quantifiers

In this section, some relations are listed among the 17 different generalised quantifiers used in the FraCaS test set by Cooper [1996]: *'a / an', 'every', 'some', 'all', 'each', 'the', 'no', 'many', 'several', 'most', 'a few', 'few', 'both', 'neither', 'one of the'*[2], *'at least CD'*[3] *and 'at most CD'*. The relations used among these quantifiers are akin to MacCartney and Manning [2008], where they used the following symbols: '$=$' for *equivalence*; '$\sqsubset$' for *forward entailment*; its converse, '$\sqsupset$' *backward entailment*; '$\curlywedge$' *exhaustive exclusion*; '$|$' *alternation* or *non-exhaustive exclusion*; '$\smile$' *cover* or *non-exclusive exhaustion*, and; '#' *independence*, which covers all

[2]Similar to 'exactly one'
[3]Which covers any cardinal number

other cases[4]. Certain research papers inspired certain relations, such as the ones carried out by MacCartney and Manning [2008], Icard III [2012], Vendler [1962], Kayne [2007], Poesio [1994] and Barwise and Cooper [1981]. For instance, MacCartney and Manning [2008] said: *“Among generalised quantifiers, we find that all = every, every $\sqsubset$ some, some $\wedge$ no, no | every, at least four $\smile$ at most six, and most # ten or more”.* According to Icard III [2012], no $\sqsubset$ not every, no | every, some $\wedge$ no. In Poesio [1994], the author agreed with VanLehn [1978] when assuming that the determiner *‘the’* followed by a plural noun will be equal to the following quantifiers: *‘all’, ‘every’* and *‘each’*. Barwise and Cooper [1981] stated *many* $\sqsubset$ *some* and *a few* $\sqsubset$ *some*. However, if there is a hierarchical tree for these quantifiers, as in WordNet hypernym relations, it would be very likely to have the particles *‘all’, ‘every’ and ‘each’* as roots of that tree, since they subsume all the remaining particles, except the negation items (i.e., *‘No’* and *‘Neither’*). The particle *‘each’* is considered as one of the roots, according to Vendler [1962], who explained the equivalence relation between the particles *‘all’* and *‘each’* (as seen in the two following sentence examples 9 and 10):

(9) *All those blocks are yellow.*

(10) *Each of those blocks are yellow.*

The particle *‘most’* is smaller than the roots and larger than other words, such as *‘many’, ‘several’, ‘a few’,... etc.*. Formally, *‘most’* is interpreted by Westerståhl [2011] as the following equation:

$$most_M(A, B) \Leftrightarrow \mid A \cap B \mid > \mid A - B \mid \tag{4.8}$$

Equation 4.8 is correct if the meaning of *‘most’* is ‘more than half’, rather than ‘a large majority’, which is a more appropriate definition. As there is an element of vagueness involved in this particle, we decided to deem it a root that is equal to the particles *‘all’, ‘every’* and *‘each’* - unless there is a fact that denies this. For instance, if we have a rule that says *‘Most swans are white’* and a fact *‘X is a swan’*, then we can consider this to be enough information to assume that *‘X is white’*, unless we have a fact denying this assumption, e.g., *‘X is black’*. Nevertheless, there is no specific size for the domain of any of these roots; consequently, MacCartney and Manning [2008] set a (#) relation between *‘most’* and *‘ten or more’*. Similarly, there should be no relation between ‘most’ and quantifiers with cardinal numbers (e.g., *‘at least five’*, *‘at most ten’* and *‘both’*[5]). As these roots belong to one category, aside from some cases regarding *‘most’*, each relation with any other quantifier should be the same (e.g., *‘every’* | *‘no’*) and should be applied to all remaining root-level members. The determiners *‘a / an’* and *‘one of the’* are located at the lowest level of the tree, as they are subsumed by every other member in the collection, except the negations. Although *‘One of the’* contains a CD, it is easily handled,

[4]These symbols are going to be used henceforth

[5]As both is referring to a CD which is ‘2’

as its formal representation is equal to the representation of *'a / an'*, as in Equation 4.9:

$$OneOfThe_M(A, B) \Leftrightarrow \mid A \cap B \mid = 1 \tag{4.9}$$

However, this is not valid if $\mid A \cap B \mid > 1$. Regarding *'at least CD'* and *'at most CD'*, they can only be compared to the low-level determiners and with each other (as the domain size does not matter in this instance). For example, the relation that MacCartney and Manning [2008] stated, *'at least four'* $\smile$ *'at most six'*, it can be clarified from the two following Equations 4.10 and 4.11.

$$AtLeastFour_M(A, B) \;\Leftrightarrow\; \mid A \cap B \mid \; \geq 4 \tag{4.10}$$

$$AtMostSix_M(A, B) \;\Leftrightarrow\; \mid A \cap B \mid \; \leq 6 \tag{4.11}$$

Between the top and bottom of the tree, there is a middle class that contains much ambiguity among its members, which are *'some', 'many', 'several', 'a few'* and 'few'. Moreover, it is considered that the sentence *'the man is running'* subsumes the sentence *'a man is running'*, while the subsumption does not hold up in the opposite direction; consequently, *'the'* $\sqsubset$ *'a'*. By building relations upon these notes, we can conclude the set of relations presented in Table 4.1. As can be seen from Table 4.1, there are some relations between generalised quantifiers. The relation between the *hypernyms* and their specific instances *hyponyms* are considered an asymmetric relation. If we extract the top three instances of a hypernymy relation from the table, we have: $every \sqsubset A$, $some \sqsubset A$ and $some \sqsupset every$. As an instance, the examples 11 and 13 are instances of true relations, as the actual relation between the quantifiers used in them is a subsuming relation $\sqsubset$; however, this will not be true if we try to conclude the TexTailment on the opposite side (see examples 12 and 14). Yet, as the relation between 'some' and 'every' is subsumed by (i.e., $some \sqsupset every$), it would not be possible to get a successful match in Example 15, while it is also possible on the other side; see Example 16:

(11) *Every footballer is rich* $\approx$ *A footballer is rich*

(12) *A footballer is rich* $\not\approx$ *Every footballer is rich*

(13) *Some footballers are rich* $\approx$ *A footballer is rich*

(14) *A footballer is rich* $\not\approx$ *Some footballers are rich*

(15) *Some footballers are rich* $\not\approx$ *Every footballer is rich*

(16) *Every footballer is rich* $\approx$ *Some footballers are rich*

	A / An	Every	Some	All	Each	The$_{+plural}$	No	Many	Several	Most	A few	Few	Both	Neither	One of the	At least four	At most six
A / An	=																
Every	⊏	=															
Some	⊏	⊐	=														
All	⊏	=	⊏	=													
Each	⊏	=	⊏	=	=												
The$_{+plural}$	⊏	=	⊏	=	=	=											
No	⋏	|	⋏	|	|	⋏	=										
Many	⊏	⊐	⊏	⊐	⊐	⊐	⋏	=									
Several	⊏	⊐	#	⊐	⊐	⊐	⋏	#	=								
Most	⊏	⊐	⊏	⊐	⊐	⊐	|	⊏	⊏	=							
A few	⊏	⊐	⊏	⊐	⊐	⊐	⋏	=	=	⊐	=						
Few	⊏	⊐	=	⊐	⊐	⊐	⋏	⊐	⊐	⊐	⊐	=					
Both	⊏	#	#	#	#	#	|	#	#	#	#	#	=				
Neither	⋏	|	⋏	|	|	⋏	=	⋏	⋏	|	⋏	⋏	|	=			
One of the	=	⊐	⊐	⊐	⊐	⊐	⋏	⊐	⊐	⊐	⊐	⊐	⊐	⋏	=		
At least four	⊐	#	#	#	#	#	#	#	#	#	#	#	#	#	#	=	
At most six	⊐	#	#	#	#	#	#	#	#	#	#	#	#	#	#	⌣	=

Table 4.1: Relations among some generalised quantifiers (DetHyps)

4.2.1.4 Polarity marking

As the approximate matching process will match each subtree in tree *A* with its parallel subtree in tree *B*, some syntactically matched subtrees were found not to be matched from a semantic perspective. That is because they may occur in an opposite monotone, as Example 17 shows:

(17) *T: I know that he will come.*
H: I doubt that he will come.

In Example 17, there is only a single difference between *T* and *H* that requires only one edit between these two sentences: substituting *'doubt'* with *'know'*, which is equal to 1 in the Levenshtein distance [Levenshtein, 1966] (as mentioned in Section 2.3.3). As seen, the context of the *T* sentence is considered as a positive context or an upward (↑) monotone, while sentence *H* has a negative context or a downward (↓) monotone. Therefore, akin to the NatLog system of MacCartney and Manning [2007], our system considers upward and downward monotonicity before checking the TexTailment between any parallel nodes. This is done by giving all the nodes a positive mark by default, i.e., '+', which denotes *upward monotone*. Therefore, a sentence such as *T* in Example 17 is parsed using our python reimplementation of the basic algorithm underlying MaltParser (mentioned in Section 3.4), to get the following tree:

```
[word(know, VB, position=1), [word(i, PR, position=0)], [word(come, VB, position
=5), [word(that, TH, position=2)], [word(he, PR, position=3)], [word(will, MD,
position=4)]]]
```

Then, after applying some post processing by keeping only the word labels and marking them, the output of the parser is converted into the following list and its' tree is seen in Figure 4.9:

```
[(know, +), [(I, +)], [(come, +), [(that, +)], [(he, +)], [(will, +)]]]
```

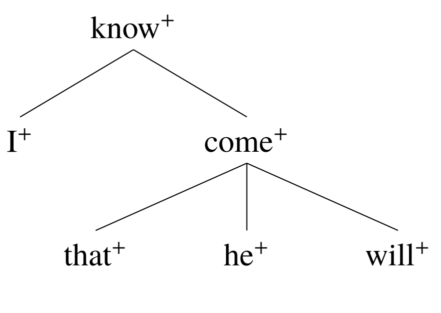

Figure 4.9: Monotonicity-twisters free example

When checking the sentence *H* in Example 17, which has been judged as having a negative context, it was found that there is one word *'doubt'* that affects the context from being positive to negative. There are also similar words which may reflect the meaning of a context, such as: *'dislike'* and *'disagree'*. Consequently, upon spotting any of these words in any given sentence, we convert the polarity marking for the remainder of that sentence. Thus, a sentence such as *H* in Example 17 is converted into the following list, where symbol '-' denotes *downward monotone* (see its tree in Figure 4.10):

```
[(doubt,+), [(I,-)], [(come,-), [(that,-)], [(he,-)], [(will,-)]]]
```

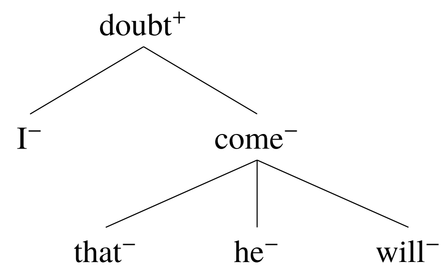

Figure 4.10: Single monotonicity twister example

Upon applying the feature of marking polarity on all given sentences, the matching algorithm in the proving system is going to take one more step of checking the term's polarities. Hence, suppose that we have three verbs α, σ and γ, where σ is a hyponym verb of α and γ is an antonym of α, we have six cases as follows:

1. If we have α in a positive context (i.e., α^+) and we would like to unify it with the similar verb α in a positive context (i.e., α^+), then the unification holds. For example, `(love, +)` $\approx$ `(love, +) = True`, as we have similar words and similar polarity markings. However, this status applies only if both verbs have the same marking, even in a negative context, i.e., `(love, -)` $\approx$ `(love, -) = True`

2. If we have α in a positive context (i.e., α^+) and we would like to unify it with the similar verb α in a negative context (i.e., α^-), then the unification does not hold. For example, `(love, +)` $\approx$ `(love, -) = False`, as although we have similar words, we have different polarity markings, regardless of which mark each side obtains.

3. If we have α in a positive context (i.e., α^+) and would like to unify it with the other verb σ in a positive context (i.e., σ^+), then the unification holds. For example, `(love, +)` $\approx$ `(like, +) = True`, as we have an approximate matching between verbs using wordNet hypernym relations, and more importantly, both words have similar polarity markings. Care should be taken, `(love, -)` $\approx$ `(like, -) = False`, as the use of

hypernym-hyponym is asymmetric; however, it holds if the opposite was, for instance, `(like, -) ≈ (love, -) = True`

4. If we have α in a positive context (i.e., α^+) and we would like to unify it with the other verb σ in a negative context (i.e., σ^-), then the unification does not hold. For example, `(love, +) ≈ (like, -) = False`, as although we have a hypernym-hyponym relation between the two words, they have different polarity markings.

5. If we have α in a positive context (i.e., α^+) and we would like to unify it with the other verb γ in a positive context (i.e., γ^+), then the unification does not hold. For example, `(love, +) ≈ (hate, +) = False`, as there is no hypernym-hyponym relation between these two words, even though they have similar positive polarity markings.

6. If we have α in a positive context (i.e., α^+) and we would like to unify it with the other verb γ in a negative context (i.e., γ^-), then the unification holds. For example, `(love, +) ≈ (hate, -) = True`, as they have opposite meanings and opposite polarity markings. Note: as both opposing words have different polarity markings, the entailment holds, regardless of which word is located in the positive context.

What we mean by antonyms is the complementary antonyms, e.g., (*alive* and *dead*), (*dry* and *wet*), and (*sick* and *well*). We do not cover graded antonyms, such as: (*black* and *white*) and (*good* and *bad*), as if something is not good, it does not mean that it is bad.

4.2.1.5 Negations

In the previous section, we introduced examples for certain words that reflect the polarity markings, as they convert the meaning of a sentence. Mainly, the negation words *'no', 'not'* and *'neither'* have the same effect, as they convert the meaning too. Changing polarity markings for dependent words functions successfully with verb twisters, such as *'doubt'*; however, the negation words, e.g., *'not', 'no'* do not have dependents in our dependency tree. Therefore, we considered promoting them to the top of their current subtree (i.e., to act as roots of their heads), so that we are sure the job of changing markings will apply to all dependents (see Example 4.12). Another feature these negations have is that they can occur in a sentence that already contains a polarity twister item; therefore, they might exist in either a positive or a negative format. The action that we take in this case is akin to one of the seven rules for CNF, which is $\neg\neg\phi \Rightarrow \phi$.

In the case of having more than a single member of the monotonicity twisters in one sentence, then we twist the marking twice, once for each occasion that a member exists, Example 18 clarifies this point.

(18) *I have no doubt that he will come*

The marking for Example 18 can be distributed as follows, with Figure 4.11 illustrating its tree

```
[(have,+), [(I,+)], [(doubt,+), [(no,-)], [(come,-), [(that,-)],
                    [(he,-)], [(will,-)]]]]
```

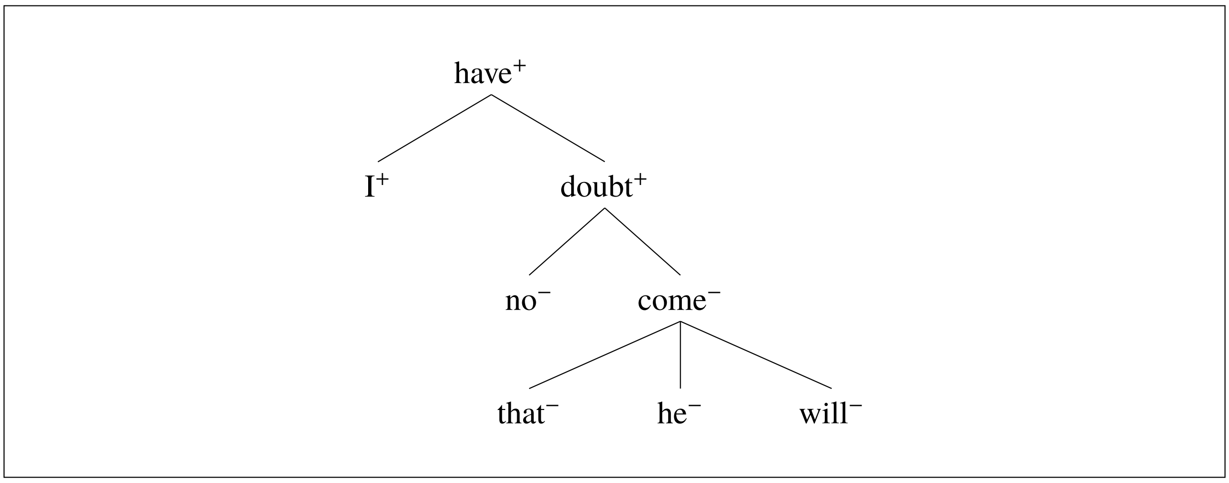

Figure 4.11: Double monotonicity twisters example

As seen in Figure 4.11, the phrase *'that he will come'* should be positive from the context, but it is marked as negative. This is clearly because the negation word *'no'* is affected by its parent *'doubt'*, but it does not have daughters to reflect their markings. Consequently, we took an action by promoting the negation words, e.g., *'not'* to be the head of its current head, i.e., 'doubt', and that is going to fix the issue of the wrong marking in this tree (see Figure 4.12).

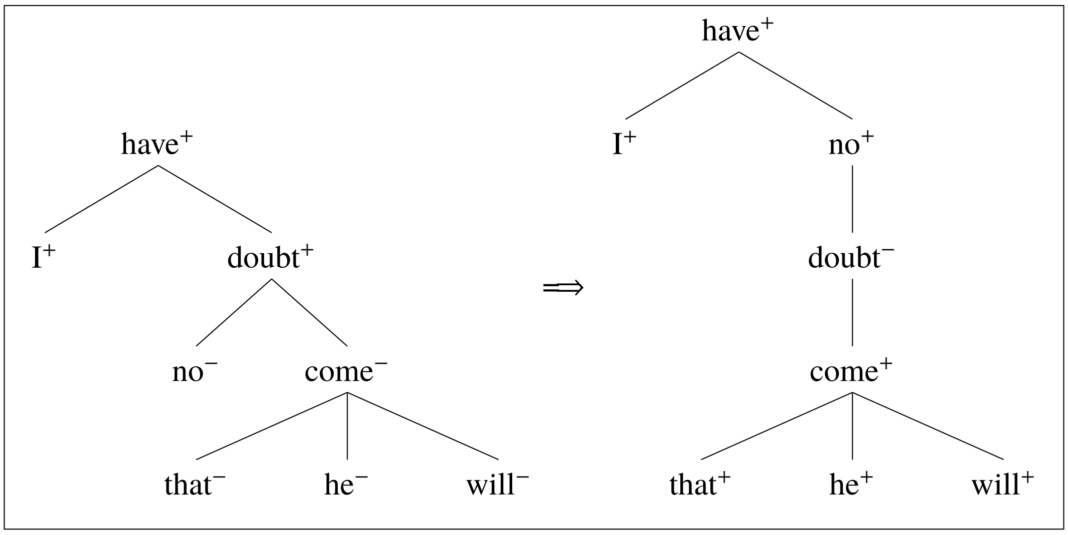

Figure 4.12: Promoting negation words

As seen in Figure 4.12, the negation word is promoted to be the head of its head. Although we have done the markings, we do not have a rule to deal with negations yet. For instance, if we have examples that contain antonyms, e.g., *'alive'* and *'dead'* (as in Examples (19) and (20)):

(19) *He is dead*

(20) *He is not alive*

and we begin comparing nodes for these trees, we will not be able to handle the negation word *‘not’*. Therefore, we have decided to delete the negation words after they take their right place in the tree and reflect their dependencies. For instance, Figure 4.13 will show how we delete the negation word that we saw earlier in Figure 4.12.

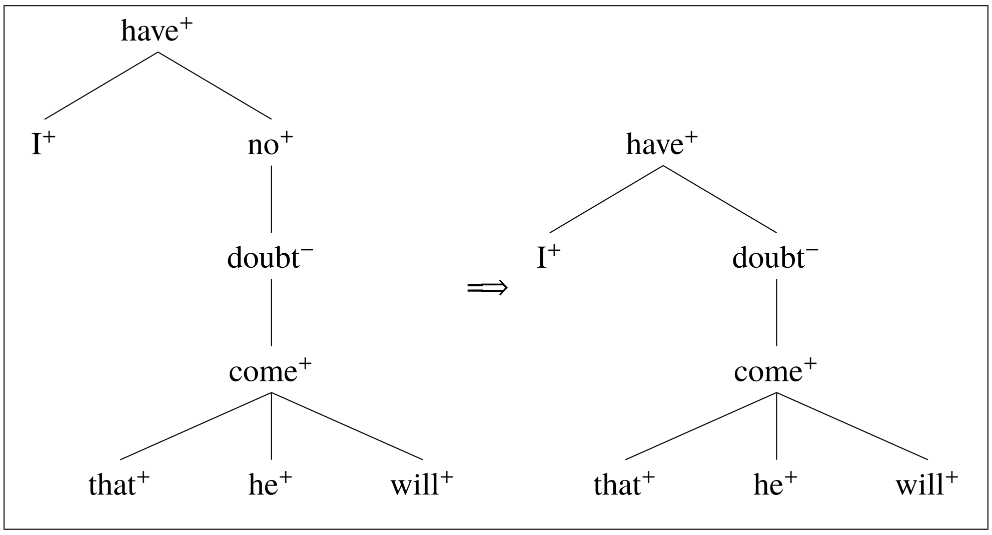

Figure 4.13: Removing negation words 1

After removing the negations, we can deduce the entailment between Examples 19 and 20 (see Figure 4.14)

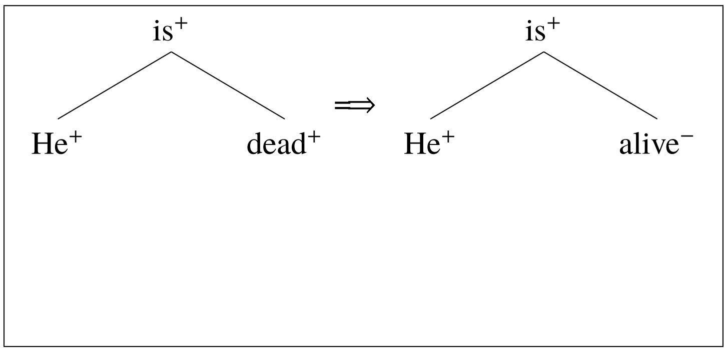

Figure 4.14: Removing negation words 2

As seen in Figure 4.14, TexTailment holds between the two sentences: the first two nodes, i.e., ‘He’ and ‘is’ are similar in both the markings and the labels, while the last parallel nodes contain two antonyms with different markings, e.g., ‘dead$^+$’ and ‘alive$^-$’.

4.2.1.6 Variables

An important part of constructing generic inference rules is the use of variables, for instance, `'X is aged → X is very old'` (see Chapter 5). Because of the importance of these variables, our theorem prover is designed to cope with them. We initiated a class for the variables and each variable has a value. This value might be a single word (as seen in Example 21), a subtree that contains many nodes (see Example 22) or it might be another value (see Example 23).

(21) *I saw a cat in the park ≈ I saw a (*`X`*) in the park*

(22) *I saw* (`X`) ≈ *I saw a cat in the park*

(23) *I saw (X) ≈ I saw (*`X`*)*

Example (23) differs from (21) and (22) in that it contains variables on both sides, i.e., in the fact and the hypothesis, which is like the example of a rule given above. The variable in this example can take any possible object, such as: *'a cat in the park', 'a pen', 'something',* etc., in which it must be bound to the same value on both ends. When the system attempts to do unification between two nodes where one of them is a variable, it firstly ensures that the variable is free and does not bind to any value, where the empty value is denoted as (???) in our system. Variables are bound in long chains, and when uniting, we dereference the elements before carrying out the unification. When the dereference of the variable is equal to '???', that implies it is has not been bound to any value.

4.2.2 Backward chaining

To conduct reasoning between a text T and a hypothesis H, theorem provers employ sets of background knowledge, i.e., inference rules, in order to find a match. These rules are connecting two different things: LHS and RHS. If one of the rules has the text T as its LHS and the hypothesis H as its RHS, then the theorem prover can find the proof. There are two ways that a proof system can go: (i) **forward chaining**, from the text to the hypothesis and (ii) **backward chaining** from the hypothesis to the text. Suppose that we have the information detailed in Figure 4.15, we would have one fact, three rules and one goal. In the forward-chaining algorithm, the proof works from existing facts and it will work forward using existing rules to derive the goal. This algorithm can be repeated multiple times, so that it can be logically described as a repeated application of modus ponens (see Equation 2.2 in Section 2.2.2).
On the other hand, the backward chaining algorithm takes, as its name suggests, the opposite direction to the forward-chaining, as it works backward from the goal and chains through the existing rules to derive the fact. The forward-chaining algorithm in Figure 4.15 would begin with the fact. If the fact matched the goal, then we would have an entailment relation; otherwise, it will look for any rule with an antecedent that matches the fact, before trying to find a relation

between the consequence of that rule with the goal. As seen in Figure 4.15, there are three suitable rules that give facts as their antecedents. One rule may lead the theorem prover to the desired goal; however, it is very likely that, after trying all these rules, the goal is not obtained. For that reason, theorem provers use the opposite direction, in which they usually begin with the goal and then move back towards the fact. As we always have a goal, i.e., a hypothesis, then it is easier to go from the goal towards the text, especially if the number of rules is not small. When our theorem prover would like to find a proof between *T* and *H*, it takes *H* and sees if it exists in the facts; if so, then there is a proof. If the goal does not exist amongst the facts, then the approximate matching algorithm tries to unify a goal with that fact; if unsuccessful, then this is the time for looking at the inference rules. From the set of rules, the prover looks for a rule where its LHS matches the goal, then it tries to prove its RHS. If the RHS can be proved, then we look for the next thing to be done. However, if it is not the case, then it tries to find another rule that leads to the conclusion. To clarify this point, Figure 4.16 shows a brief version of a trace of our proving algorithm, where the full version can be found in Appendix A.1. The given *T* and *H* are from Example (24) and the rules that are used in this proof are shown in Figure 4.15:

(24) *T: it is a strong effect.*
H: it is a blow

```
Fact:  [word(is, VX, position=1), [word(it, PR, position=0)], [word(a, DT,
        position=2)], [word(strong, JJ, position=3)], [word(effect, NN,
        position=4)]]
-----------------------------------------------------------------------------
Rule:  [[is:  VX, [?X: ???], [a:  DT], [strong:  JJ], [effect:  NN]]]
        ⇒
         [is:  VX [?X: ???], [impact:  NN, [an:  DT]]]

Rule:  [[is:  VX, [?X: ???], [impact:  NN, [an:  DT]]]]
        ⇒
         [is:  VX, [?X: ???], [blow:  NN, [a:  DT]]]
-----------------------------------------------------------------------------
goal:  [word(is, VX, position=1), [word(it, PR, position=0)],
         [word(blow, NN, position=3), [word(a, DT, position=2)]]]
```

Figure 4.15: Facts, rules, and a goal example

```
Does the goal exist in the facts?.
  unify(goal, fact)
  (Both are lists)
  Try to match current heads
  unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
  unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
  matched, continue
    .
    .
    .
    .
  Didn't unify word(is, VX, position=1), word(is, VX, position=1)]
prove(goal, [facts])
trying this rule:  [[is:  VX, [?X: ???], [a:  DT], [strong:  JJ], [effect:  NN]]] ==> [is:  VX, [?X: ???], [impact:  NN, [an:  DT]]]
(goal is:  <tb.SENTENCE instance at 0x14aad2638>)
unify(goal, currentRuleConsequent
  (Both are lists)
  Try to match current heads
  unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
    unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
    matched, continue
    .
    .
    .
    .
  Didn't unify word(is, VX, position=1), word(is, VX, position=1)]
prove(goal, [facts])
trying this rule:  [[is:  VX, [?X: ???], [impact:  NN, [a:  DT]]]] ==> [is:  VX, [?X: ???], [blow:  NN, [a:  DT]]]
(goal is:  <tb.SENTENCE instance at 0x14aad2638>)
unify(goal, currentRuleConsequent
  (Both are lists)
  Try to match current heads
  unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
  unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
  matched, continue
    .
    .
    .
    .
         unify([] (list), [] (list))
  Both are empty lists
```

```
Does the goal exist in the facts?
  unify(currentGoal, fact)
    .
    .
    .
    .
  Didn't unify word(is, VX, position=1), word(is, VX, position=1)]
prove(goal, [facts])
trying this rule:  [[is:  VX, [?X: ???], [a:  DT], [strong:  JJ], [effect:
 NN]]] ==> [is:  VX, [?X: ???], [impact:  NN, [a:  DT]]]
(goal is:  <tb.SENTENCE instance at 0x1445d2c20>)
unify(goal, currentRuleConsequent
  (Both are lists)
  Try to match current heads
  unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
  unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
  matched, continue
    .
    .
    .
    .
         unify([] (list), [] (list))
  Both are empty lists
Does the goal exists in the facts?
  unify(currentGoal, fact)
    .
    .
    .
    .
          unify([] (list), [] (list))
    Both are empty lists
  raise exception
  proof.SUCCESS
```

Figure 4.16: A trace of our proving algorithm

4.3 Summary of the chapter

To summarise this chapter, there are currently various theorem provers that can conclude an entailment between any two given sentences using inference rules. A problem with the provers is that they are not suitable for non-logical inputs. Consequently, our theorem prover has been built so that it can solve this issue with several other features presented in the following points:

1. It accepts inputs in a readable format (dependency trees), so that there is no need to translate the input into any logical format.

2. When trying to prove, this system is looking amongst available facts, to see whether it can find a fact that leads to the given hypothesis or not.

3. If the hypothesis does not exist among the facts, the prover tries another way to conclude a proof. It looks amongst available inference rules to see whether it can find an antecedent of a rule that leads to the desired point (fact) or not.

4. The algorithm of proving is a backward chaining algorithm, as it starts from the goal and goes to the fact; whenever a path is found, it proceeds, otherwise, it goes back to the last point.

5. To check for matching between a hypothesis and one of the antecedents, this system employs another non-deterministic unification function that tries to do approximate matching between the two trees, i.e., the hypothesis and an antecedent. This prover considers that a tree t_1 approximately matches another tree t_2 when one of the following points applies:

    (a) If there is an equivalence between t_1 and t_2, (e.g., $t_1 = t_2$)

    (b) For all nodes of t_1 (e.g., $x_1, x_2, ..., x_n$), and all nodes of t_2 (e.g., $y_1, y_2, ..., y_n$), if each parallel node (x_i, y_i) has an equivalence relation, e.g., $x_i = y_i$ or subsuming relation, e.g., $x_i \subseteq y_i$, then we can say that t_1 approximately matches t_2 or $(t_1 \approx t_2)$.

    (c) For each parallel node of t_1 subtrees (e.g., $x_1, x_2, ..., x_n$), and t_2 subtrees (e.g., $y_1, y_2, ..., y_n$), if they approximately match each other, having a remaining node that represents an adjective modifier in t_1, then the prover is allowed to drop it and we can say that t_1 approximately matches t_2 or $(t_1 \approx t_2)$. Conversely, in a negative context, the opposite action is allowed, as it is allowed for adjective modifiers in t_2 to be dropped.

    (d) When $t_2 \approx (t_1 + \text{an adjunct } d)$ in a positive context, then the adjunct can be dropped and there is an approximate match between these two sentences.

    (e) As a converse of the previous point, when $t_1 \approx (t_2 + \text{an adjunct } d)$ in a negative context, then the adjunct can be dropped and there is an approximate match between these two sentences.

6. To determine the TexTailing direction, each node (word) in each tree is given a polarity marking. All words are given (+) by default, unless some collection of words (e.g., *'no'*, *'doubt'*) exists in the context, as they have an inverted mark (-) for themselves and affect all the following nodes. When two of the words occur in one sentence, the polarity marking is reflected twice; therefore, it would be positive again for and after the second appearance of a reflecting word.

7. The previous point of polarity marking is used to decide which direction the TexTailment process should go (i.e., monotonicity). For two trees t_1 and t_2, in the case of a positive monotone/context, the TexTailing direction is $t_1 \Rightarrow t_2$; otherwise $t_2 \Rightarrow t_1$ and, as mentioned earlier, there might be more than one direction in the same TexTailing process.

8. As with most theorem provers, this theorem prover uses a backward chaining algorithm when TexTailing, as it starts from hypothesis H. By using given rules, it then tries to reach the text T. When successful, it announces that there is an entailment; otherwise, there is no TexTailment relation.

The background knowledge that is used in this system (the inference rules) was created for this system. In the next chapter, Chapter 5, there is a discussion about these inference rules.

Chapter 5

Generating inference rules

5.1 Overview

This chapter discusses the inference rules that have been used in Figure 4.15 as help tools to deduce entailment. In our theorem prover, we employed a finite set of rules with two sides: an antecedent (LHS) and a consequent (RHS) split by a right arrow (⇒). The rules used in that example were the commonsense knowledge that we relied upon to successfully conduct the proof. As mentioned in the background chapter, building commonsense knowledge is not a new task to this field, as there were several examples of these knowledge bases that contain useful information to be used in inference systems. Nonetheless, the knowledge bases mentioned do not suit the needs of this system for various reasons. For instance, we want the commonsense knowledge in this system to be definitional rules, while most of the representations of existing knowledge are relational rules, such as: 'Is-A', 'PropertyOf',... etc. Some examples of these commonsense knowledge bases that do not match the required type of representation of this system are: ConceptNet [Liu and Singh, 2004], OpenCyc [Lenat, 1995], ThoughtTreasure [Mueller, 1998] and YAGO [Suchanek et al., 2007]. OMCS contains facts written as raw English sentences; however, what our system needs is a collection of rules, not simple facts. Moreover, OMCS and YAGO use people to participate in feeding in their data by encouraging them to play word games, such as 'Verbosity'. Although permitting the public to contribute to databases aids to enlarge these databases, it may affect their credibility, as there are no restrictions upon contributors' backgrounds and expertise. On the other hand, the DIRT algorithm by Lin and Pantel [2001] may be useful, as it generates inference rules written in natural language; however, from the results presented in Table 2.1, there are fewer than 50% paths leading to correct rules; thus, we are trying to gain more accurate rule databases.

Therefore, we attempted to create a commonsense knowledge base that has the following two characteristics: (i) The database must contain knowledge as rules not facts; (ii) They must be written in raw English texts that are easily parsed, and; (iii) The rules used in this commonsense knowledge base should be guaranteed to hold true information. One of the best available sources to achieve this task is English dictionaries; these dictionaries contain definitions in which the nature of each definition is a relation between a word (definiendum) and its meaning (definiens). As well as containing definitional information, dictionaries tend to use short and clear sentences written with faultless grammar. Additionally, dictionaries can be guaranteed to have, at least, some precise information about every word in the language.

The rules used in the previous chapter's Figure 4.15 are considered commonsense knowledge that a system relies on to conduct inferences. The rules were extracted from an online dictionary *The Macmillan Dictionary* (TMDC), in a process of multiple steps to reach a point of having inference rules concerning the form `LHS` $\rightarrow$ `RHS`. In the remaining sections of this chapter, there is a description of the steps that have been taken to build natural language and commonsense knowledge using an English dictionary.

5.2 The Macmillan Dictionary

A major advantage of language dictionaries is that they offer definitions for each word in a particular language. These definitions may be written differently from one dictionary to another, in terms of length and complexity; however, we will aim to use shorter definitions to make clear and concise inference rules. An appropriate choice for this task was The Macmillan Dictionary (TMDC), as it usually utilises short and patterned sentences in its definitions, when compared to other well-known dictionaries, such as: Oxford[1] and Cambridge[2]. Table 5.1 presents the first definition of a randomly chosen set of common open-class words; a couple of examples have been taken from each class. Although shorter definitions sometimes affect the precise meaning, they deliver meaningful sentences and are an appropriate choice for such a project. In addition to it containing simple form definitions, The Macmillan Dictionary contains a set of 7,500 marked words that are believed to denote 93% of everyday English words. This set of words was used as the scope for this experiment during the current phase. This collection of words is not separately available, neither in the main site of the dictionary nor in its API; therefore, we began to collect them, the method for which is discussed in Section 5.2.1.

[1]https://en.oxforddictionaries.com
[2]http://dictionary.cambridge.org

Open class	Word	Some English Dictionaries		
		Oxford	Cambridge	Macmillan
Noun	Time	The indefinite continued progress of existence and events in the past, present, and future regarded as a whole.	the part of existence that is measured in minutes, days, years, etc., or this process considered as a whole.	the quantity that you measure using a clock.
	Person	A human being regarded as an individual.	a man, woman, or child.	an individual human, usually an adult.
Verb	Say	Utter words so as to convey information, an opinion, a feeling or intention, or an instruction.	to pronounce words or sounds, to express a thought, opinion, or suggestion, or to state a fact or instruction.	to express something using words.
	Get	Come to have (something); receive.	to obtain, buy, or earn something.	to obtain, receive, or be given something.
Adjective	Good	To be desired or approved of.	very satisfactory, enjoyable, pleasant, or interesting.	of a high quality or standard.
	New	Produced, introduced, or discovered recently or now for the first time; not existing before.	recently created or having started to exist recently.	recently created, built, invented, or planned.
Adverb	Up	Towards a higher place or position.	towards a higher position; towards a higher value, number, or level.	in or towards a higher position.
	Just	Exactly.	now, very soon, or very recently.	used for saying when something happens.

Table 5.1: Comparison of dictionaries

5.2.1 Words collection

With a lack of choices to glean all the important words into an exclusive list, the only available method to check if a word is among the TMDC list of commonly used English words is to send a request to TMDC, looking up the goal word in the dialogue box and seeing whether it returns as red in colour or not. This strategy would be acceptable if the search targeted a short list of known words, which is not the case at this point, as the target words are unknown and their quantity is around 7,000 words.
Entering a huge amount of words to check whether they are red-coloured or not is a non-trivial task, as it would cost a vast amount of time and effort. Therefore, a tool was created to collect all the words by taking Wiktionary's top 100,000[3] most frequently used English words[4] and entering each word into the dictionary to confirm whether it— the word (*w*)—is one of our targets or not, dependent on the dictionary feedback. When TMDC replied to the request, it transformed the tool into the page of the requested word. When *w* has more than one tag, each of its tags feature a separate page in TMDC; therefore, when looking up words of this kind, TMDC will make a decision to transform the request to an available page, not all of them. The suggested site that The Macmillan Dictionary refers to is the most used version of the goal word, meaning that if it is not red-coloured, the other variants are not within the commonly used words either. The technique that this tool follows to decide whether the given word is among our targets is to use the source code for the page that TMDC refers to, and to seek if that source page contains the class `<h1 class="redword">` or not. Whenever this tool finds that a given word is not red-coloured, then it removes that word from the huge Wiktionary list; thus, we were able to extract most of the red-coloured words in a separate list of 6,735 words.

5.2.2 Definitions extraction

The output of the previous phase in Section 5.2.1 is the basis of the current experiment, as the definitions being investigated are the definitions of only the extracted words. The output of the definition extraction will be saved into a nested JSON[5] dictionary file, in which the outer level of the dictionary has the goal words as keys and their values are inner dictionaries; their keys are the POS of the keyword (w_{pos}) and all its' definitions (w_{def}).
To pinpoint the values of w_{pos} and w_{def} in the source page of the determined word, two functions are released with a request to the appropriate classes in which their values are located. For the w_{pos}, they can be found in a class called `<span class="PART-OF-SPEECH">`. The importance of getting the POS with the retrieved definition lies in the following phase of rewriting definitions (in Section 5.2.4). In this regard, words may have more than one tag; for example, the word

[3]If Wiktionary does not list all the tenses and forms for every word, then it is expected that the top 10,000 is enough to do this step.
[4]https://en.wiktionary.org/wiki/Wiktionary:Frequency_listsTop_English_words_lists
[5]JavaScript Object Notation

‘back’ comes as an adverb; e.g., *‘Do not look back’*, an adjective; e.g., *‘The back garden’*, a noun; e.g., *‘He has broken his back’* and a verb *‘We will back you all the way on this one’*. Each of word the versions of *back* has its own page in the dictionary that contains a definition and examples. If the noun version of the word ‘back’ is given as red, it does imply that the adjective ‘back’ is red. However, despite this not being the case here, there are cases concerning different words, such as the noun ‘knot’ being red-coloured, while the verb ‘knot’ is not. Moreover, another issue exists in which there are words with a list of all possible tags in a single page; such words make it difficult to decide what is the most appropriate tag to assign it. However, it is not common to have these cases, as the total of these words is only 129 words—representing only 2% of the total size—so they were excluded from the data collection. An instance of this problem can be seen on the page of the word *‘outside’*, as it has been classified as [adjective, adverb, noun, preposition] at the same time.

The definitions are trickier to extract than the tags as—in most cases—they contain more than one word. With the issue not only being the length, some of the words within the definitions are actually words that have separate pages themselves in the dictionary and are therefore covered under linking tags. Consequently, to extract these definitions, two sub-functions work together to aggregate the definition words. The first sub-function specifies the range of the definition from the beginning of the definition class `<span class="DEFINITION">` until its end `</span>.*?($|<div class="EXAMPLES"`, where the `"EXAMPLES"` class always follows the definition. When the range is specified, the second sub-function goes within the borders and extracts all the words that it contains, either linked or non-linked words. For example, one of the definitions of the noun *‘human’* in TMDC is *‘a person’*; the word *‘person’* is a clickable word with a link to its own page in the dictionary. Its page link can be seen in Figure 5.1. The piece of code in Figure 5.1 clearly shows how definition words are listed in the source page, as both cases of words are seen in this figure.

```
<span class="DEFINITION" resource="dict_british"><span class="SEP

DEFINITION-before"> </span>a <a href="http://www.macmillandictionary.com

/dictionary/british/person" class="QUERY" title="person">person</a></span>

<div class="EXAMPLES"...
```

Figure 5.1: A portion of a definition source page from The Macmillan Dictionary

After running this crawling system for all given words, we found that these important words create 12 different classes (see Table 5.2).

It can be seen from Table 5.2 that there are four categories that represent more than 98% of the total words; unsurprisingly, these are the open-class word categories (i.e., 1,283 adjectives, 335 adverbs, 3,447 nouns, and 1,431 verbs). Note that the verbs refer to lexical verbs, as

No	Category	Count	Example
1	abbreviation	1	etc: *used after a list of things to mean 'and others of the same type', when you do not want to mention everything.*
2	adjective	1283	illegal: *not allowed by the law.*
3	adverb	335	painfully: *in a painful way.*
4	conjunction	16	if: *used for introducing a situation that may happen or may be real, especially when talking about its results.*
5	determiner	10	our: *belonging to or connected with you and the group that you are a part of, when you are the person speaking or writing.*
6	interjection	12	please: *used as a polite way of asking for something or of asking someone to do something.*
7	modal verb	10	will: *used for saying what you expect to happen in the future.*
8	noun	3447	win: *an instance of winning, especially in sports.*
9	number	6	billion: *the number 1,000,000,000*
10	preposition	22	during: *at one point within a period of time or a process.*
11	pronoun	33	nobody: *no one*
12	verb	1431	note: *to notice or realise something.*
total		6606	

Table 5.2: The 12 categories in The Macmillan Dictionary

auxiliary verbs have different tags. These four types are the only tags that will be targeted in our commonsense knowledge base; as a consequence, the remaining tags were filtered out of the experiment.

5.2.3 Definiens patterns

As seen in the previous section, there is a JSON dictionary that contains words with their features. Following each definition, there is a *'tagged_definition'* list that splits definition words, and tag each of them. So the tagged definition is a big list that contains inner lists, in which each inner list contains two items: a word and its POS. To get these tags, we used a *'Brill tagger'* [Brill, 1992] with the *'Maximum likelihood tagger'* as an underlying tagger that are trained on the English treebank from the Universal Dependency Treebank (UDT) [McDonald et al., 2013]. The form of this list is `[word`$_1$`, tag], [word`$_2$`, tag], ..., [word`$_n$`, tag]`, where n is the last word in a given definition; for instance, `[['a', 'DT'], ['person', 'NN']`. The tag set in the UDT is slightly different than the tag set for the Penn Treebank, as UDT does not use special tags for plural nouns, past verbs, ... etc. It uses very generic sets, e.g., 'NN' for nouns, 'VB' for verbs, 'RB' for adverbs and 'JJ' for adjectives. To reproduce these definitions as rules, we want to know how these definitions structured and to know how these definitions are structured, we tried to get their patterns. Having the benefit of the UDT tag sets, we could extract generic patterns at this stage. A definition's pattern represents the tags used in that definition in their correct order; therefore, the pattern for *'a person'* is `'DT NN'`. The process of checking the patterns of all the collected definitions is carried out by reading the entire JSON dictionary extracted from the last step, then extracting the tags from each definition and sending them to their appropriate repositories. Four repositories are generated—one for each tag—to collect their definition patterns. For all patterns added, there is a counter to tally how many times each pattern occurred, so that we are aware of how most of the definitions are constructed. The counter is used to avoid keeping some strangely structured definitions, therefore we only keep the patterns that are used in at least five definitions. Below each added pattern, there are some definitions taken from the dictionary to decide how such patterns should be rewritten in the following step. See Figure 5.2 for a short list; a more extensive list for adjective words is provided in Appendix B.

```
“RB JJ” = 88
aged:  very old
alternative:  not traditional
amazing:  very surprising
ancient:  very old

“JJ” = 41
clear:  transparent
comic:  funny
cross:  angry
dear:  expensive
```

Figure 5.2: A portion of the most frequent definitions' patterns for adjective words in TMDC

Note that dictionaries—in general—contain words with several definitions, the most generic of which is placed at the top of the definitions' lists. However, in the process of extraction, the location of the definition in the list was not considered, as the tool sought the definitions for similar patterns. For example, the adjective *'alternative'* has the first definition in the dictionary as *"different from something else and able to be used instead of it"*, whereas the second definition suits our needs better, as it contains only two words (see Figure 5.2).

It was notable that some patterns were used very frequently; for example, among noun class definitions, the `[DT NN]` pattern was used more than 80 times, e.g., *advocate: a lawyer*[6], a similar number of times for the pattern `[DT NN NN]`. Among the verbs, each of the patterns `[TO VB DT NN]` and `[TO VB DT NN]` were used for more than 65 examples, for example, *conquer: to win a victory*[7]. Amongst the adjective definitions, the most repeated pattern is `[RB JJ]`, which is used in 88 definitions; Figure 5.2 demonstrates this with some examples. Amongst the adverbs, it is common to have one-word definitions from the class `RB`, such as *absolutely: completely*[8], which is used in more than 40 definitions.

[6]http://www.macmillandictionary.com/dictionary/british/advocate_2
[7]http://www.macmillandictionary.com/dictionary/british/conquer
[8]http://www.macmillandictionary.com/dictionary/british/absolutely

In general, it was observed that patterns that begin with `(RB)` are very common among adjective definition patterns, while among the noun patterns, the determiners `(DT)` were the greater part. For verbs, the majority began with `(TO)` and two tags shared most of the adverb pattern heads: `(RB)` and `(IN)`. From that observation, a post-processing step was taken by grouping patterns depending on their heads, i.e., their first elements. This move was taken to create generic patterns out of the existing ones, using regular expressions (see Section 5.2.4). Therefore, the ordering used in Figure 5.2 changed so that all `(RB)` patterns came before `(JJ)`. The new order is presented as `(“RB JJ”)` then `(“RB VB”), (“RB JJ CC JJ”), (“RB JJ CC NN”), (“RB NN”)` respectively, followed by the `(JJ)` and its group.

5.2.4 Rewriting definitions

The form of the extracted definition obtained from the last section is not enough to create rules. Therefore, this section will illustrate the process of turning a relation of `(word : definition)` into a complete sentence that can be easily parsed.

Pre-processing

For all extracted patterns, we set a counter that reads inside the obtained JSON dictionary and returns how many definitions match their instances. When the function catches a definition that fits one of the patterns, it then rewrites its words and their tags, i.e., its tagged definition, in the form of `‘FORM!!TAG’`, leaving white spaces between definition words. For instance, the definition *‘advocate: a lawyer’* is generated as: `‘a!!DT lawyer!!NN’`, *‘precise: strictly correct’* as `‘strictly!!RB correct!!JJ’` and so on. The double exclamation marks are used so that the system is aware that the items on both sides of them are linked.

Rewriting process

The main goal for the entire experiment is to generate inference rules, where each side of a rule is represented as a parsable sentence that can be used by the theorem prover described in the previous chapter; therefore, the role of this step is to convert dictionary definitions into parsable sentences. The main idea is to transfer each word (w) and its definition pattern (d_p) to be in a form such as `'X is` w `if X is` d_p’. For instance, the pattern of the definition for the word *‘precise’* is `‘strictly!!RB correct!!JJ’` and is converted into `(X is precise if X is strictly correct)`, where `X` is going to be constructed as a variable when transforming this text into inference rules. This is, however, not the only form of an output sentence, as adjectives, adverbs and verbs have quite different forms. Therefore, some hand-crafted regular expression (regex) rules were written to deal with such different cases. Using regular expressions for this task is an appropriate solution, as they are great to deal with sets of strings that conform to a specific pattern. These regular expression rules were made generic so that one rule can match many patterns with similar skeletons to reproduce them in a similar way. These generic rules are a collection of tuples, where each tuple contains three items (a POS, a regex pattern and a string). The POS refers to the classification of the definiendum and the string is the output sentence that should be generated from this pattern. In the following lines, some cases will be highlighted to illustrate how these regular expressions were constructed for nouns, verbs, adjectives and adverbs.

- **Nouns:** In nouns, from the list of patterns collected previously, it is often seen that noun definitions begin with determiners (DT). There are various patterns for noun definitions that are headed by a determiner. Some examples of these patterns include: `[DT NN]`: *'advocate: a lawyer'*, `[DT NN NN]`: *'advisory: an official warning'*, `[DT NN IN NN]`: *'age: a period of history'* and `[DT JJ NN]`: *'asset: a major benefit'*. Other instances have a different type of determiner, which is a cardinal (CD), as in the definition *'course: one of the parts of a meal'*, which has the pattern `[CD IN DT NN IN DT NN]`. In regular expressions, it can chunk the patterns into groups—thanks to the feature of the named capturing group—thus, we can explicitly define a pattern's parts and make it possible to capture any determiner, i.e., (DT) or (CD), that comes at the very beginning of a pattern by grouping them under a group called `<det>`. For noun definitions, the rule can be straightforward, as most of their extracted patterns are similar; hence, the rule can be something such as `("noun", "^(?P<det>\S*!!(DT|CD)?)\s*(?P<MOD>(\S*!!(IN|JJ|NN|DT)\s*)*)$",`
  `"X is a %s`
  `if X is \g<det> \g<MOD>")`, where %s refers to the definiendum word and `!!` is taken from the d_p that we created previously. In case the article that precedes the (%s) word is 'an' not 'a', then this can be fixed by using libraries such as *'en'* from NodeBox[9], which can take inputs as `en.noun.article("university")` to return `'a university'` and `en.noun.article("hour")` to produce `'an hour'.` By using this pattern, the above examples can be turned into strings as: `'X is an advisory if X an official warning', 'X is an age if X is a period of history', 'X is an asset if X is a major benefit'`, and `'X is a course if X is one of the parts of a meal'.` Moreover, similar patterns exist without a determiner and they have the same output as in *'bread: money'* and *'agony: great pain'* so that the determiner group is created to be optional, as indicated by the mark `"?"` that follows the list of determiners in the pattern afforded to this group.

[9]https://www.nodebox.net/code/index.php/Linguistics

- **Adjectives:** In general, the extracted examples and patterns for adjective definitions begin with (RB). For instance, the most used pattern is `[RB JJ]`, e.g., *'alternative : not traditional'*, followed by `[RB JJ CC JJ]`, e.g., *'appalling : very unpleasant and shocking'* and `[RB VB]`, e.g., *'awake : not sleeping'*. However, these adjective definitions can be rewritten in a similar way to the nouns, as it is observed that they are similar to the noun definition that have no articles in their heads. Therefore, when constructing the output, we use a rule such as `("adjective", "^(?P<MOD>(\S*!!(IN|JJ|NN|DT)\s*)*)$", "X is %s if X is \g<MOD>")`. This rule should rewrite the previous examples to sentences like: *'X is alternative if X is not traditional', 'X is appalling if X is very unpleasant and shocking'*, and *'X is awake if X is not sleeping'*.

- **Verbs:** These can be different than the previous couple of categories, as verbs may exist in different types. A verb is called *an intransitive verb* if it does not take an object, e.g., *'He ran'*, or *a transitive verb* when the verb requires an object, e.g., *'He drives a bus for a living'*, and *a ditransitive verb* when the verb requires two objects *'Maureen gave Dan the pencil'*.
  Per TMDC definitions, there are tags for transitive and intransitive verbs, where 'transitive' class verbs also involve ditransitives. Besides, some verbs are of a hybrid class [transitive/intransitive] as in : *'give: to do something good or helpful for someone'*. Thus, there is an issue that these classes that are provided by the TMDC are not enough to know whether there is an object, or an object and an indirect object or indeed anything reliable about transitivity. To face this problem, we looked at the verb definitions for direct objects to conclude two distinct forms of objects.

  An object might occur as a particular word that delivers a special meaning to the definition, such as *'cycle: to use a bicycle'*, where not many objects can replace the word 'bicycle'. Another form of direct objects occurs when they come as generic words that can take different values, as in: *'afford: to provide something'*. The general object word 'something' is substituted with any noun; for instance, *'afford a car'* means *'to provide a car.'* Since generic terms, such as: *'something'*, *'someone'* and *'somebody'* can take different values, we thought that they could be turned into variables so that within the inference process they could be occupied with various objects. Some definitions contain a couple of generic terms, as in ditransitive verbs, e.g., *'give: to pass something to someone'*, whereas transitive verbs may be linked by 'or', as in: *'assist: to help someone or something'*. For definitions that have a single generic term, or double terms linked by 'or', we replace their objects by the variable `(Y)` according to the following rule, where the construction for the right-hand part is illustrated in Figure 5.3, and str_1? and str_2? denoting optional piece of texts:

`("verb", "^(?P<to>\S*!!TO?)\s*(?P<MV>verb)(?P<REST1>word*?)(?P<OBJ>SOMEONE`[10]`)`
`\s*(?P<REST2>word*?)$", "X %s Y if X \g<MV> \g<REST1> Y \g<REST2>")`. This rule fits transitive verbs with definitions that begin with the word ‘to’, followed by a verb then the rest of the definition, where the rest of the definition contains a generic term that is denoted in the rule by ‘SOMEONE’. The third part of the rule shows the final form of the output as `(X afford Y if X provide Y)`.

$$\begin{tikzcd}
\textit{To} & \textit{main-verb} \arrow[rrrrr, bend left] & \textit{str1?} \arrow[rrrrr, bend left] & \textit{generic-term} \arrow[rrrrr, bend left] & \textit{str2?} \arrow[rrrrr, bend left] & \Rightarrow & \textit{X main-verb} & \textit{str1?} & \textit{Y} & \textit{str2?}
\end{tikzcd}$$

Figure 5.3: Rewriting transitive verb definitions

- **Adverbs:** Generally, the extracted adverb definitions have been found to be very short. There are many cases in which the definition contains only one word, such as *‘commonly: usually’*. There are also cases when the definition has more than one word, such as *‘always: on every occasion’*. An adverb is, in general, used to modify verbs, adjectives or other adverbs; therefore, we introduced variables to all of its definitions. One-word definitions can take the variable just after the definition, so the example above can be turned into: `(X commonly Y if X usually Y)` according to this rule:
  `("r", "^(?P<RB>\S*!!RB?)\s*$", "X %s Y if X \g<RB> Y")`
  In case the definition has more than one word, it is often that the definition begins with a preposition (IN), so we add an auxiliary verb (does) followed by the variable (Y) in front of that preposition, e.g., `(X always Y if X does Y on every occasion)`, according to the following rule:
  `("r", "^(?P<IN>\S*!!IN?)\s*(?P<MOD>(\S*!!(DT|JJ)\s*)*)(?P<HD>nn*)`
  `(?P<POSTMOD>(\S*!!IN\s*nn\s*)*)$", "X does Y %s if X does Y \g<IN>`
  `\g<MOD> \g<HD> \g<POSTMOD>")`

5.2.5 Transforming to inference rules

Since the inference process depends on inference rules, not simple English sentences, this step involves transferring the obtained sentences from the last step into inference rules. All generated

[10]SOMEONE is a variable that covers the existence of *‘something’*, *‘someone’*, and *‘somebody’*.

sentences have a form where there is a main-clause followed by an if-clause, where each clause has a shared variable, such as: X, Y or them both. By taking one of the examples mentioned earlier, *'X is awake if X is not sleeping'*, both clauses share the same labelled variable (X). If a main-clause precedes an if-clause, a reader can understand it as whenever the if-clause is true, the main-clause is true too. This has a similar notion to the Horn clause inference rules, where for each rule `A ⇒ B`, whenever A is true, then so is B. Inspired from this notion, we converted all of these sentences into rules, where each rule has an antecedent in the left-hand side (*LHS*) and a consequent in the right-hand side (*RHS*) in the form of *LHS* ⇒ *RHS*. The consequences of the inference rules are occupied by the main-clauses of the sentences, while if-clauses—excluding the word 'if' at the beginning—represent the antecedents of the rules. As the left-hand side of the rules contain single items, the generated rules are considered to be Horn clauses. When applying these regular expressions on the collected TMDC definitions, the machine could extract 4,613 inference rules. Most extracted rules were produced by the noun rule, and we were not surprised by that, as most tokens of words are nouns, hence, most types should be nouns as well. Nouns represent 62.9% of the total generated inference rules, followed by verbs, which represent 28.6%. Far from these figures, the adjectives rules produced only 6.6%, and the lowest portion was for adverbs, which represent only 1.8% of the total rules. To check the validity of the used patterns, we attempted to apply them on different data, which are the WordNet definitions.

5.3 WordNet definitions

In WordNet, there is a library incorporating the definitions of most of the words it contains. From our collection of words in the last experiment, we could find definitions for approximately 90% of them, as there are quite a few words with empty definitions. WordNet contains lists of synsets; thus, when looking for `wordnet.synsets("back")`, the result will be a list of items consisting of the type synsets, as follows:

```
[Synset('back.n.01'), Synset('rear.n.05'), Synset('back.n.03'), Synset('back.n.
04'), Synset('spinal_column.n.01'), Synset('binding.n.05'), Synset('back.n.07'),
Synset('back.n.08'), Synset('back.n.09'), Synset('back.v.01'), Synset('back.v.
02'), ...  etc].
```

According to Fellbaum [1998], there are 117,000 synsets in WordNet. Each of these synsets are linked together, depending on their relational concept, and this explains why we can find labels other than 'back' in the given list of the 'back' synsets above. As they fall into the same synset list, we attempted to extract all definitions for the synsets of the word 'back'. When looking for synsets of a given word, as we have written above, the results will contain more than one classification for the word, i.e., there is a noun-back, a verb-back, an adjective-back and an adverb-back. Therefore, the class of the requested word is an important factor that should be considered when making a request to gain a list of synsets. So, it is important to specify the wanted tag of the word, and that can be obtained by saying `wordnet.synsets("back", "n")`.

```
[Synset('back.n.01'), Synset('rear.n.05'), Synset('back.n.03'), Synset('back.n.
04'), Synset('spinal_column.n.01'), Synset('binding.n.05'), Synset('back.n.07'),
Synset('back.n.08'), Synset('back.n.09')].
```

The i_{th} synset definition can be obtained by calling the definition function on that synset, for instance, we do `wordnet.synsets("back", "n")[0].definition()`, to get the definition for the first synset, which is: *'the posterior part of a human (or animal) body from the neck to the end of the spine'*. Following this method, we passed the full list of the collected words and investigated whether they have any synsets in any of the four open-classes (n, v, a, r), which denote nouns, verbs, adjectives and adverbs, respectively. All definitions gained from WordNet definitions—known as WDFS henceforth—are stored in another JSON dictionary file. Like the output of TMDC, the keys of the JSON dictionary file are the labels for the collected words, and each word is a dictionary itself that has up to four keys, for the words that have synsets on all four tags. Each tag of a word has a list of definitions for the synsets of that word with that tag. These definitions are written as the previous dictionary in the form (`FORM!!TAG`)[11], so that they will be normalised and be used in the same set of regular expression patterns.
When running these definitions on the set of regular expression rules, the system could extract 3,630 inference rules. Amongst these inference rules, many were for nouns, as they represent 56.63% of the total rules, while adjectives represented 31.18% and 10.50% were adverbs. The most surprising result is that the minority were verbs, representing just 1.69%, whereas in TMDC they approximately represent a third of the total domain size. The main reason that these definitions were not caught is that the single pattern written for the verbs is restricted to the definitions that begin with 'to', followed by a main verb and containing the generic objects 'something', 'someone' and 'somebody'. In WDFS, it is notable that they use determiners at the beginning of verb definitions sometimes and they do not have the main verb right at the beginning of the definition. For instance, *'give: the elasticity of something that can be stretched and returns to its original length'*.

[11]Using a similar tagger to the previous experiment; i.e., a Brill tagger

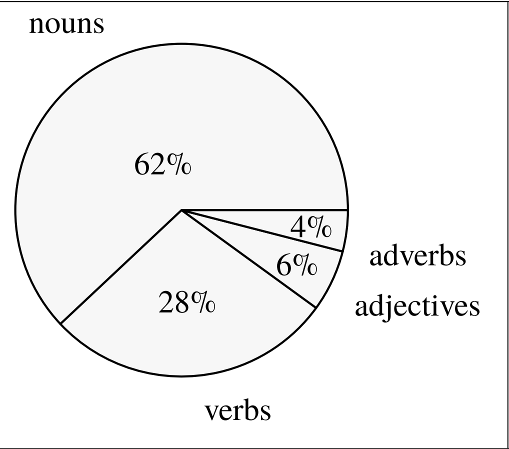	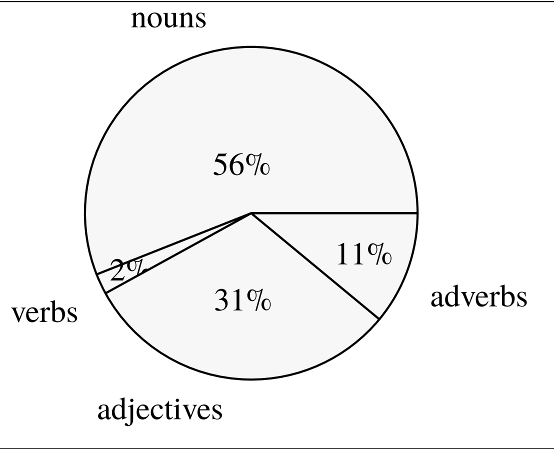

Table 5.3: The percentage of samples for each open-class from TMDC (left) and WDFS (right)

5.4 Dictionaries' rules results

At this stage, we have approximately 8,000 inference rules from both sources: TMDC and WDFS. These rules must be judged to see how accurate they are in the suggested outputs. To evaluate these rules, we took a random sample of 200 inference rules; these examples were taken equally from both TMDC and WDFS. These randomly selected examples consider how many examples each pattern catches when taking the sample. The number of examples that every tag receives is illustrated in Table 5.3. In this table, the left pie chart represents the samples taken from TMDC, whereas the right pie chart shows the percentages taken from WDFS. It is revealed from the right figure that there was a sharp drop in the number of verb examples extracted from WDFS, when compared to the ones taken from TMDC. The nouns are always taking the lead, while the other two parts, i.e., adjectives and adverbs, have more instances in the WDFS sample.

We developed a web page especially for this task, where we can upload the rules in comma-separated values format (CSV), and we shared this page with native English speakers who evaluated our rules. The participants were asked to see if these rules are correct across both sides: meaning and grammar. With each rule, there are three options: if a referee believes this rule is correct, then the 'yes' option is clicked; if he thinks that the rule is incorrect the 'no' option is chosen; otherwise, there is a skipping option if it is hard for the referee to decide to click 'yes' or 'no'. As the number of rules is quite large for an online survey, we made the order of rules dynamic so that the rule with the smallest number of answers comes first. Additionally, since we are seeking the same number of answers for each question, we control them by removing the rules from the survey whenever it reaches a certain amount of answers. In our experiment, four answers were chosen for each rule, and whenever a rule receives 75% 'yes' answers, it is considered to be true. Out of the 200 sentences, there are 148 rules that are true. From the survey results, we have calculated precision and recall using the Equations [5.1] and [5.2], respectively:

$$Precision = \frac{R}{100} \tag{5.1}$$

$$Recall = \frac{R}{100} \times \frac{M}{N} \tag{5.2}$$

Where N represents the number of definitions that we were looking at, and M denotes the number of definitions that were chosen as being potential rules. The constant (100) denotes the sample domain size and R means inference rules that were judged 'True'. From the previous equations, we got 74.0% precision, with a recall of 0.68.

If we look for the results for each dictionary separately, the precision of the rules obtained from TMDC are slightly better than the ones taken from WDFS, as they were true in 74% cases, compared to 73% in WDFS (see Table 5.4).

	TMDC	WDFS	TMDC + WDFS
Nouns	69.3%	76.7%	72.8%
Verbs	78.5%	100.0%	80.0%
Adjectives	83.3%	67.7%	70.2%
Adverbs	100.0%	72.7%	80.0%
Total	74.0%	73.0%	74.0%

Table 5.4: TMDC & WDFS samples' precision

From Table 5.4, two samples were afforded 100.0% precision: the verbs in WDFS and adverbs in TMDC. This is because we used a tight pattern for them, meaning that while we got only a small number of examples, they were all good, especially for verb definitions in WDFS. Figures 5.4 and 5.5 illustrate the charts for the number of the domain sizes for each tag with the number of correctly judged inference rules.

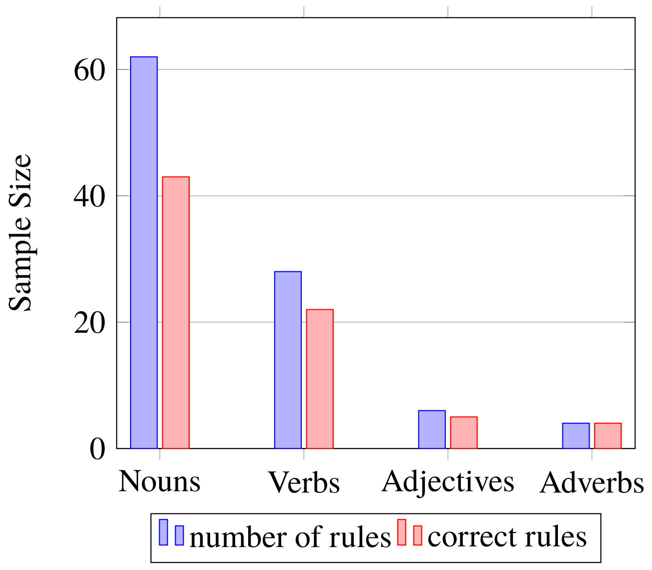

Figure 5.4: Number of True inference rules in TMDC sample

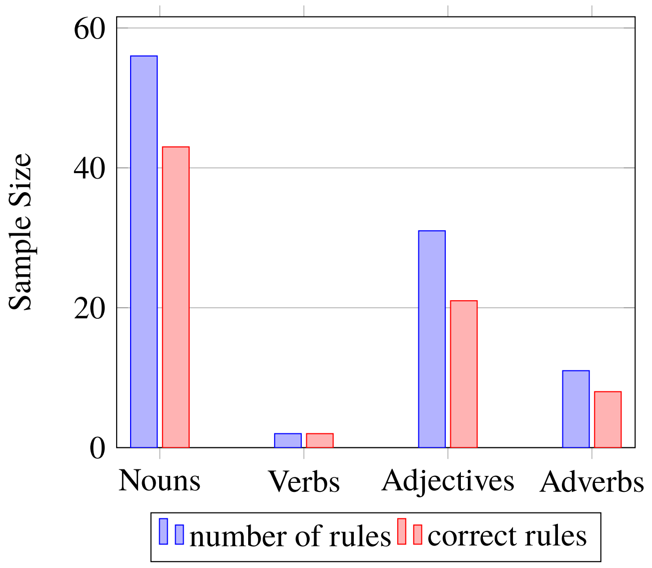

Figure 5.5: Number of True inference rules in WDFS sample

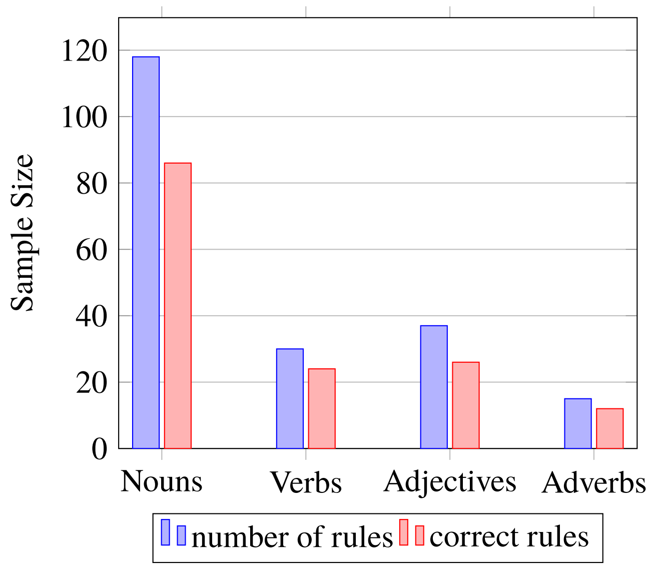

Figure 5.6: Number of True inference rules in TMDC & WDFS samples

As seen in the hybrid sample in Figure 5.6, all four tags gain approximately one non-correct inference rule in every four rules. The best record concerns the verbs, as we were very selective when deciding which rule patterns we were interested in. Nonetheless, most verb samples were taken from TMDC, as verb definitions with generic objects are not often found amongst WDFS definitions. Some issues concerning the extraction of verb inference rules were due to some generic objects occurring within intransitive definitions; for instance, *'come: to start doing something'* as in *'The new changes will come into effect next month'*. In addition, some verb definitions lack a preposition, such as: '`X intend or hope to achieve Y` → `X aim Y`', should be '`X aim to do Y`' and '`X agree to do Y` → `X consent Y`' should be '`X consent to do Y`'. Moreover, some definitions are informal, so these ones have not been accepted, for example, '`X defeat Y completely` → `X own Y`'.

Adverbs are also correct on most occasions, while there are no obvious reasons for the rules that failed. Although noun definitions have the biggest portion of inference rules, their precision is still high, where nearly three-quarters of the cases are correct. A notable issue with the noun definitions is the use of informal definitions; e.g., '`X is a prison` → `X is a can`' or old-fashioned ones; e.g., '`X is the highest seats in a theatre` → `X is a god`' or both of the two previous issues; e.g., '`X is the drug heroin` → `X is a horse`'. The decrease in precision for the adjective rules is coming from the WFDS side, as there are some definitions that seem to not make sense, such as: '`X is literal meanings` → `X is low`', '`X is in excellent physical condition` → `X is better`' and '`X is delayed` → `X is latest`'. These examples, however, do not affect the whole precision for the adjectives, neither in WFDS nor in the hybrid data test.

Despite the minor number of rules extracted in this phase of the experiment, we believe that the proposed system has strong potential. One future task of this chapter is to extract more definitions with similar patterns to enlarge and refine this commonsense knowledge base by taking some suggested steps, as follows. Refining some minor mistakes affected the precision when some sentences were generated ungrammatically, hence this should be considered, in order to get a more precise output. The words with a multi-class problem will be investigated in the near future, especially if all of the multi-classes fall into our class range of interest. In addition, some other dictionaries may suggest a variety of short definitions that are different from the ones in the existing collection.

5.4.1 Syllogistic rules

Another possible method to seek data that can be transformed into inference rules is to employ sentences from entailment test sets. If there are two sentences (p and h) and they contain shared contents, these shared contents can be exchanged with the variables. This notion is like the Aristotle syllogistic rules of ancient times, as seen in Table 5.5, where there are three snippets: two t's and one h. In the left column of the table there is an example with the shared word, while the right column has a copy of this example that has had its shared contents replaced with variables.

Example	Rule
All men are mortal	*All X are Y*
All Greeks are men	*All Z are X*
―――――――	―――――――
All Greeks are mortal	*All Z are Y*

Table 5.5: An example for Aristotle syllogistic rules

The example featured in Table 5.5 is close to the form of FraCaS two-premise problems, as it contains two facts and one goal, with some shared content. Therefore, we will take the two-premise problems of the FraCaS test as a case study in this experiment, going on to make syllogistic inference rules for them. From the multi-premise problems of FraCaS, we chose only the two-premise problems, and this is because they represent nearly 80% of the multiple premise cases. In addition, the same method that is followed to deduce a goal from two facts can be applied to n number of facts. Example 5.3 is the FraCaS problem number (013) that illustrates how it is similar to Aristotle's, where the facts are the sentences (a) and (b) and the goal is the sentence (c).

(5.3) `fracas-013`

a. *'Both leading tenors are excellent.'*

b. *'Leading tenors who are excellent are indispensable.'*

---

c. *'Both leading tenors are indispensable.'*

In Example 5.4.1, the shared words were as follows: *'both'* is shared between (a) and (c); *'leading'* is shared between (a), (b) and (c); *'tenors'* is shared between (a), (b) and (c); *'excellent'* is shared between (a) and (b); *'are'* is shared between (a), (b) and (c) and *'indispensable'* is shared between (b) and (c). If these identical words are replaced with variables, then it may be that they are generic and able to be used with different examples. By replacing open-class shared words with variables, we received Example 5.4, which has similar patterns to the rule stated in Table 5.5.

(5.4) `pattern-013`

a. *'Both X are Y.'*

b. *'X who are Y are Z.'*

---

c. *'Both X are Z.'*

A rule made out of the generic pattern in 5.4 can be used to prove the problem in Example 5.3, as well as many other similar examples. For instance, a goal as *'both businessmen are successful'* and two facts: *'both businessmen are smart'* and *'businessmen who are smart are successful'* can be solved by that pattern. Although these patterns are carefully generated from given examples, they are very generic, so they might lead to wrong proofs. To make them more precise, we put a constraint on the class of the variables' heads. For instance, a variable with the tag 'NN' can be bound to *'man'*, *'fat man'*, and *'big fat man'*, as the parse trees for all of these phrases are headed by a noun (NN). As a consequence, we construct the variable for the phrase `[word(tenors, NN, position=1), [word(leading, JJ, position=0)]]` as follows: `[word(VAR(X, ???), NN)]`.

From these given patterns, the rules are constructed where the last sentence in the pattern is a consequent and the others are antecedents. Rules have been made for each two-premise problem that has 'yes' as an answer in the FraCaS test suite. The number of problems with the answer 'yes' amongst the two-premise problems totalled 74. Constructing these rules is similar to the process of constructing the background knowledge rules in Chapter 5; the difference here is that there is more than one antecedent used in one proof; therefore, both of the antecedents are joined into one list. The form of a rule is $\texttt{[premise}_1\texttt{, premise}_2\texttt{]} \Rightarrow \texttt{antecedent}$. The form

```
[[are:  VX, [?X: NN, [both:  DT]], [?Y: JJ]], [are:  VX, [?X: NN],
[who:  WP], [are:  VX, [?Y: JJ], [?Z: JJ]]]]
⇒
[[are:  VX, [?X: NN, [both:  DT]], [?Z: JJ]]
```

Figure 5.7: Multi-premise FraCaS problems' rule

of the inference rule, if we apply it on the pattern in Example 5.4, is shown in Figure 5.7 and Table 5.6.

Premise 1	Premise 2	=>	hypothesis
[are:VX [?Y:NN [both:DT] [?X:JJ]] [?Z:JJ]]	[?Y:NN [?X:JJ] [are:VX [who:WP] [?Z:JJ] [are:VX[?A:JJ]]]]	⇒	[are:VX [?Y:NN [both:DT] [?X:JJ]] [?A:JJ]]

Table 5.6: The dependency tree for a multi-premise FraCaS rule

5.5 Summary of the chapter

This chapter presented a discussion concerning commonsense knowledge, which can be defined as the lexical knowledge any fluent English speaker should have. This knowledge is essential for inference systems, as it provides tools that help proving systems to derive a match between two items if they have a relation. Our system also requires these rules, as what we have is an inference engine that employs inference rules to deduce a proof. In the literature, there are several databases that provide these kinds of rules, such as: OpenCyc [Lenat, 1995], ThoughtTreasure [Mueller, 1998], OMCS [Singh et al., 2002], ConceptNet [Liu and Singh, 2004], and YAGO [Suchanek et al., 2007]. However, these examples do not suit the needs of our system, due to two problems in their construction. On the one hand, several of these commonsense knowledge bases contain logical annotations or are written in non-standard English sentences. On the other hand, some employ public contributors through word games, which may affect the accuracy of the entered information. Therefore, we generated our collection of inference rules (commonsense knowledge) to match our required format, as they were obtained in raw English form. This commonsense knowledge is derived from The Macmillan Dictionary; hence, we avoided nonsense-entered facts, as information in dictionaries is written by specialised linguists. Another benefit of using a dictionary is that it contains at least some information about any word in the language. On the contrary, employing word games leads to a lack of some facts concerning low-level information, e.g., the relationship between *'eating'* and *'opening the mouth'*).

This commonsense knowledge was collected by extracting definition patterns from a collection of the most common open-class words (according to TMDC). When we extracted the definitions from TMDC, it was found that there are several definitions constructed in the same way. Therefore, some regular expression patterns are written to catch similarly constructed definitions and the definitions are rewritten in parsable sentences, rather than the form of (word:definition). There was a rule or two for each tag that reads the pattern of a given definition, before chunking its parts, then reproducing it, after linking the word with its definition in a form such as `'X is <word> if X is <definition>'`. These sentences are transformed into inference rules by taking the <word> to be the consequence in the RHS, with its definitions as antecedents in the LHS. We could extract approximately 4,600 inference rules from a limited number of rules, then we attempted to test these rules upon another data set. These regular expression patterns were tested on WordNet definitions and more than 3,500 inference rules were extracted. To check the precision of these inference rules, four native English speakers were asked to identify whether they found these inference rules acceptable or not. The test was a sample of 200 rules, where 100 rules are taken from each data set. When 75% of the opinions stated that the inference rules were valid, it was then considered valid. The precision received from this experiment is 74%. The recall was 0.68 for the TMDC, but it was very low for WDFS because our set of patterns was very tight.

After that, another type of inference rules was extracted, i.e., 'syllogistic rules'. These rules were extracted from FraCaS two-premise problems and these ones have been chosen because of two reasons: (i) they are extracted from examples that proved to be true (from the FraCaS test suite), (ii) they contain shared content, so that we can turn them into variables. These kinds of rules are going to be used in solving some FraCaS problems in the next chapter.

Chapter 6

Evaluation

In the previous chapters, a discussion has been presented detailing the two non-deterministic algorithms, namely: the proving algorithm and the approximate matching algorithm (see Chapter 4). Then, some inference rules were generated from two different English dictionaries and FraCaS test suite problems (see Chapter 5). In this chapter, we are going to test the system on the FraCaS test suite. Since the approximate matching algorithm is prepared to do certain tasks, we will focus only on the results we gleaned from the related topics as 'quantifiers'.

6.1 FraCaS test suite

The FraCaS test suite [Cooper et al., 1996] was a project carried out by the FraCaS Consortium[1] and it is one of the available test sets for TexTailment systems. It involves a collection of problems, with each problem having one premise (or more) and a question. Each of the sentences involved are considered to be comparatively simple sentences which, in most cases, are like one another. Thus, ensuring the process of transferring a sentence from p to h is possible with the application of a small number of edits. Regardless of this simplicity, these problems cover a broad range of semantic and inferential phenomena; therefore, it is considered to be a suitable choice to test our system.

[1]A large collaboration in the mid-1990s, aimed at developing a range of resources related to computational semantics

?P	Topic	No.	Sentences	Answer
1	Quantifiers	46	p: *Neither commissioner spends time at home.*	NO
			h: *One of the commissioners spends a lot of time at home.*	
2	Quantifiers	13	p1: Both leading tenors are excellent.	YES
			p2: Leading tenors who are excellent are indispensable.	
			h: Both leading tenors are indispensable.	

Table 6.1: Random single-premise inference problems from the FraCaS test suite

Number of premises	*Number of problems*	*Percentage*
1	192	55.5%
2	122	35.3%
3	29	8.4%
4	2	0.6%
5	1	0.3%

Table 6.2: Distribution of FraCaS premises

6.1.1 FraCaS characteristics:

The number of problems contained in the FraCaS test suite is 346. Each problem has one or more premise sentences followed by a question, and in most cases, an answer.
Table 6.1 shows a couple of examples that are taken from Tables 1.1 and 1.2 in the introduction chapter.

- ***Single and multiple premises:*** The FraCaS test suite contains 346 problems. There are 192 single-premise problems that represent approximately 55.5% of the total domain sizes. More about the quantity of premises is shown in Table 6.2.

  Only the first and second premise problems are included in the evaluation in Section 6.1.2. We took the problems with two premises to illustrate that, unlike shallow inference systems, this system has the ability to do something with multi-premise problems.

- ***Problem hypothesis:*** Most problems in the FraCaS test suite have questions; each question stands as a goal of the problem that it belongs to. As the aim of TexTailment is to determine the inferential validity of a premise and a hypothesis, MacCartney and Manning [2007] converted each FraCaS question into a declarative hypothesis. This conversion process was done automatically by generating a syntactic parse for each question, before applying some rules to transform the resulting trees before they annotated the outcomes manually, should they contain any grammatical mistakes. Fortunately, the authors have made their results freely available [2], so we were able to obtain the hypothesis for all problems with a question.

- ***Answer types:*** In most cases, the answer for FraCaS problems takes one of three options, namely, *YES, NO, UNK*, while only a small amount of problems do not fall into these

[2]http://nlp.stanford.edu/ wcmac/downloads/fracas.xml

Sequence	*Topic*	*Number of problems*	*Percentage*
1	`Quantifiers`	80	23%
2	`Plurals`	33	10%
3	`Anaphora`	28	8%
4	`Ellipsis`	55	16%
5	`Adjectives`	23	7%
6	`Comparatives`	31	9%
7	`Temporal`	75	22%
8	`Verbs`	8	2%
9	`Attitudes`	13	4%

Table 6.4: The nine topics in the FraCaS test suite

categories. Table 6.3 illustrates how many times each answer occurs. The type *'other'* is a collection of answers that contain qualifiers, such as *'Yes, on one possible reading'* in problem `(fracas-087)`. In addition, some other cases have ambiguous answers, such as *'not many'* in problem `(fracas-012)` and *'Yes on one scoping; unknown on another scoping'* in problem `(fracas-308)`. Moreover, there are four problems without answers, as they have been written without questions; therefore, no hypothesis and, consequently, no answers. Like the action taken by MacCartney and Manning [2007], we have sidestepped all of the *'other'* problems in our evaluation process. This system generates binary-classifications for the answers where it combines the *NO* and *UNK* answers.

Number of answers	*Answer type*	*Percentage*
180	YES	52%
94	UNK	27%
31	NO	9%
41	OTHER	12%

Table 6.3: Distribution of FraCaS answers

- ***Topics:*** The problems in the FraCaS test suite are designed to cover nine different sections. These sections, and how many examples each topic has, are shown in Table 6.4.

6.1.2 System testing

To evaluate this system on FraCaS, we have two types of problems and two types of algorithms. The first problem type is the single-premise problem, for which examples can be solved by

utilising the approximate matching algorithm only (see Section 6.1.2.1). The second type of problem is the multi-premise problem, which requires the use of inference rules (as seen in Section 6.1.2.2). After attempting these tests, we will extend the set of problems by aiming to solve problems that require the use of both algorithms (see Section 6.1.2.3).
When using these rules, exclusively made for these examples, we were able to get 100% of the problems right. Another reason this test was not expected to fail is because the shared contents are equivalent to each other and the trees are similar, regardless of how they were constructed.

6.1.2.1 Approximate matching test

As we are only employing the approximate matching algorithm at this stage, we cannot solve anything other than the single-premise problems. The problems with the single premise are spread out across all nine topics. Since we are only focusing on some lexicon relations related to generalised quantifier relations and skipping modifiers, we are only going to try the related topics. The most related topics were 'quantifiers' and 'adjectives', while the others require aspects that we do not cover, even though a few of those problems are solvable with the existing algorithm. To unify p and h, the matching algorithm requires the dependency trees to be constructed correctly. Hence, for a problem (x) if the premise's tree x_p is constructed differently from the hypothesis tree x_h, then that problem will be terminated from the test. Each of the three matching techniques are controllable techniques, which means they can be enabled and disabled separately. This feature gives us an indication of which techniques are playing a more important rule amongst the targeted problems; then, all these techniques will be switched on simultaneously.
Amongst the 192 single-premise problems, there are nine problems with an 'undefinite' answer, so they were excluded. MacCartney and Manning [2007] labelled some answers as 'undefinite' if they lack either a hypothesis or a well-defined answer. From the remaining 183 problems, we extracted a number of problems for which we can glean the right tree; the number of extracted problems totalled 70. By excluding the non-related topics, the number of problems was reduced to 28, and that is our test's domain size. When this system proves a problem, it is marked as a 'yes'; otherwise both 'no' and 'unknown' answers are marked as 'no'. Table 6.5 details the number of problems that our system observed, showing how many times the system received a right answer. The four columns under the approximate matching table cell illustrate the status of each technique in each of the four tests. The first test is carried out by switching the word relations on (denoted in the table as: σ), with switching the other techniques off; i.e., detHyps (γ) and skipping modifiers (δ). The last line shows how many right answers can be obtained when all three techniques are switched on.

Topic	Number of problems	Approximate matching techniques			
		$\sigma = 1$ $\gamma = 0$ $\delta = 0$	$\sigma = 0$ $\gamma = 1$ $\delta = 0$	$\sigma = 0$ $\gamma = 0$ $\delta = 1$	$\sigma = 1$ $\gamma = 1$ $\delta = 1$
Quantifiers	20	12	12	19	19
Adjectives	8	5	5	6	7
Total	28	17	17	25	26

Table 6.5: First test on 28 single-premise FraCaS problems

From Table 6.5, it is obvious that one of the problems within the quantifiers problems was incorrectly guessed, and that problem was `(fracas-056)`, where the system can find the alignment while its gold answer is 'unknown'. The action that our system takes to decide its deduction is that it drops the adjective modifier 'British' from the premise; however, that seems not to lead toward a successful conclusion for that problem (as seen in Example 6.1).

(6.1) `fracas-056, answer: unknown`

a. *'Many British delegates obtained interesting results from the survey.'*

---

b. *'Many delegates obtained interesting results from the survey.'*

Example 6.1.2.1, however, addresses a tricky problem, as it is hard to decide which way the direction of the monotonicity should be after the quantifier 'many'. MacCartney and Manning [2007] found it dubious, as they said FraCaS people may think about determiners as denoting a proportion, while systems interpret them as an absolute number.

Baseline matching algorithm

Because our parser did not make the right tree for all of the examples that have been solved by MacCartney and Manning [2007], we cannot compare our work to what they did. To overcome this issue, we designed a baseline matching algorithm to compare its results with our matching algorithm. That baseline is an enhanced version of a string edit distance algorithm, which uses the cheapest edit operation possible to transform the text T into the hypothesis H. The baseline algorithm employs two edit operations, namely, exchange (CHN) and delete (DEL). The costs of these operations are variable, for instance, the cost of deletion is 2, while it costs nothing if

Topic	Number of problems	Accuracy (Baseline)	Accuracy (Our algorithm)
Quantifiers	20	50.0%	95.0%
Adjectives	8	62.5%	87.5%
Total	28	53.6%	92.8%

Table 6.6: Comparison between the baseline and our matching algorithms

we delete an adjective. Similarly, the cost of exchanging is 3, but when we exchange a word with one of its hyponyms in the WordNet hyponymy tree, then it costs 0. Each cost is added to a scale. Whenever the value of the scale exceeds a threshold, then no TexTailment holds between these two sentences; otherwise it holds. Applying this algorithm to the set of 28 examples in Table 6.5, we obtained the results in Table 6.6 that show a comparison between the baseline and our matching algorithms.
From the two experiments, we observe that our matching algorihtms do better than the enhanced string edit algorithm in the way that it handles polarity. For example, the baseline algorithm cannot catch the entailment in the following problem:
P: Few committee members are from southern Europe.
H: Few female committee members are from southern Europe.
Additionally, our matching algorithm addresses the relations between generalised quantifiers, while the baseline matching algorithm does not. For instance, consider the following problem:
P: A few female committee members are from Scandinavia.
H: At least a few committee members are from Scandinavia.

6.1.2.2 Multi-step proofs

This is another test where we apply our syllogistic inference rules from Chapter 5 to deduce an entailment for a selection of two-premise problems. In the first phase of this test, we will test all of the 74 inference rules on the 74 FraCaS problems that these rules have been extracted from. This test was completed without issues, as these rules are basically built on these problems. In this test, there are some sentences that are difficult to parse, but because the rules are constructed from the syntactic trees for the problems themselves, we did not find a problem.

6.1.2.3 Backtracking with approximate matching test

Generic rules:

In the first test, there was an analysis of single-premise problems concerning approximate matching techniques without using inference rules. The second test was applied on two-premise problems that require inference rules, done by reproducing FraCaS patterns as rules before using them as inference rules. This test will go a step further when the first and the second tests are applied together. Therefore, this test will be applied by using both algorithms: the backtracking and the approximate matching. This test contains two sub-tests, where both algorithms are required simultaneously. The first sub-test is to find generic rules. A generic rule is a rule that can prove more than one problem with the aid of an approximate matching algorithm. Each rule is tested on all problems. When a rule can prove more than one example, it is classed as a generic rule, and the rule for the other example is deleted from the rules list. Amongst the single-premise rules, there are several examples in FraCaS that fit one pattern, especially in the quantifier's topic. However, we are only interested in the two-premise problems, where it is less likely to find generic rules. An example of a two-premise generic rule can be found for the problems `(fracas-002)` and `(fracas-003)` in Examples 6.2 and 6.3.

(6.2) `fracas-002`

a. *'Every Italian man wants to be a great tenor.'*

b. *'Some Italian men are great tenors.'*

---

c. *'There are Italian men who want to be a great tenor.'*

(6.3) `fracas-003`

a. *'All Italian men want to be a great tenor.'*

b. *'Some Italian men are great tenors.'*

---

c. *'There are Italian men who want to be a great tenor.'*

There are 74 inference rules that can be tested in this stage, and if we do a test for all of the 74 rules, we will obtain some generic rules, as shown in Figure 6.7. Figure 6.7 shows that there are nine problems that can share four generic rules. However, not all of these rules are appropriate syllogistic rules. An appropriate syllogistic rule contains only variables and closed-class words in all of its sentences. That means that if one sentence in any of the rules contains a non-shared verb or noun, then this rule is not generic, as it contains a word that adds some meaning to the rule. Therefore, from Figure 6.7, the fourth generic rule has a pronoun 'John', which prevents this example from being an appropriate syllogistic rule. In addition, this generic rule has only open-class words, and we cannot make a rule from only variables. For these reasons, the last rule is not appropriate. Generally, it has been found that the 'quantifiers' have better examples compared to the rest of the topics, and in this table, five problems out of nine are from the quantifiers. When removing useless inference rules, i.e., ones that can be covered by generic rules, the remaining set of rules became 71 rules. For the first generic rule that have the problems (2) and (3), which are similar, apart from the quantifiers used, therefore we keep the more generic rule. In our DetHyps table, *'Every'* and *'All'* belong to the same level, so either rule can be useful, and we can delete the other. The same decision applied for the second generic rule, as *'Each'* is also from the same level. For the third generic rule, there is a difference in the sentence (P2), where it is weaker in problem (134) than in (135), therefore, the 135 rule can be used for 134 but not vice versa.

Generic Rule No.	FraCaS No.	Fracas problem	
G.R(1)	2	P1	*Every Italian man wants to be a great tenor.*
		P2	*Some Italian men are great tenors.*
		H	*There are Italian men who want to be a great tenor.*
	3	P1	*All Italian men want to be a great tenor.*
		P2	*Some Italian men are great tenors.*
		H	*There are Italian men who want to be a great tenor.*
G.R(2)	66	P1	*Every resident of the North American continent can travel freely within Europe.*
		P2	*Every Canadian resident is a resident of the North American continent.*
		H	*Every Canadian resident can travel freely within Europe.*
	67	P1	*All residents of the North American continent can travel freely within Europe.*
		P2	*Every Canadian resident is a resident of the North American continent.*
		H	*All Canadian residents can travel freely within Europe.*
	68	P1	*Each resident of the North American continent can travel freely within Europe.*
		P2	*Every Canadian resident is a resident of the North American continent.*
		H	*Each Canadian resident can travel freely within Europe.*
G.R(3)	134	P1	*Every customer who owns a computer has a service contract for it.*
		P2	*MFI is a customer that owns exactly one computer.*
		H	*MFI has a service contract for all its computers.*
	135	P1	*Every customer who owns a computer has a service contract for it.*
		P2	*MFI is a customer that owns several computers.*
		H	*MFI has a service contract for all its computers.*
G.R(4)	157	P1	*John owns a red car.*
		P2	*Bill owns a blue one.*
		H	*Bill owns a blue car.*
	159	P1	*John owns a red car.*
		P2	*Bill owns a fast one.*
		H	*Bill owns a fast car.*

Table 6.7: A list of the FraCaS examples that shared generic rules.

Extending problems:

Another set test is to be extended from the two-premise problems, where the shared elements of the sentence are not identical (A = B) but approximately match (A $\approx$ B). The extension process depends on the context of a sentence, as the inference here is directional. In positive contexts, there are four points to be considered, which are described in the following lines:

- For open-class shared-words:
  - **In the premises:** The value of the variable can be replaced with one of its hypernyms in WordNet.
  - **In the goal:** The value of the variable in the goal can be replaced with one of its hyponyms in WordNet.
- For determiner shared-words:
  - **In the premises:** The value of the variable can be replaced with an upper level determiner from detHyps hierarchical relations, if it exists (see Table 4.1).
  - **In the goal:** The value of the variable can be replaced with a lower level determiner from DetHyps hierarchical relations, if it exists (see Table 4.1).
- For adjective modifiers:
  - **In the premises:** For nouns that are not modified with adjectives, then adjective modifiers can be added to them.
  - **In the goal:** For nouns that are modified with adjectives, then adjective modifiers can be deleted from them.
- For prepositional modifiers:
  - **In the premises:** If they do not contain prepositional modifiers, then prepositional modifiers can be added to them.
  - **In the goal:** If they contain prepositional modifiers, then prepositional modifiers can be deleted from them.

In the case of negative contexts, the opposite of these points are taken; for instance, for the first point of open-class shared words, a hypernym of a word can used in the goal, not in the antecedents, and so on. Each of the above points has been applied in Table 6.8.

No.		FraCaS problems	Extension	Extended problems
099	P1	***Clients** at the demonstration were all impressed by the system's performance*	a hypernym in the facts	***Consumers** at the demonstration were all impressed by the system's performance*
	P2	*Smith was a **client** at the demonstration*		*Smith was a **consumer** at the demonstration*
	H	*Smith was impressed by the system's performance*		*Smith was impressed by the system's performance*
099	P1	*Clients at the demonstration were all impressed by the system's performance*	a hyponym in the goal	*Consumers at the demonstration were all impressed by the system's performance*
	P2	*Smith was a client at the demonstration*		*Smith was a consumer at the demonstration*
	H	*Smith was impressed by the system's **performance***		*Smith was impressed by the system's **act***
004	P1	*Every Italian man wants to be a great tenor*	an upper-level determiner in the facts	*Every Italian man wants to be a great tenor*
	P2	***Some** Italian men are great tenors*		***Many** Italian men are great tenors*
	H	*There are Italian men who want to be a great tenor*		*There are Italian men who want to be a great tenor*
131	P1	*Each department has a dedicated line*	a lower-level determiner in the goal	*Each department has a dedicated line*
	P2	*They rent them from BT*		*They rent them from BT*
	H	***Every** department rents a line from BT*		***Many** department rents a line from BT.*
208	P1	*Mickey is a small animal*	adding adj modifiers to the facts	*Mickey is a **nice** small animal*
	P2	*Dumbo is a large animal*		*Dumbo is a large animal*
	H	*Mickey is smaller than Dumbo*		*Mickey is smaller than Dumbo*
003	P1	*All Italian men want to be a great tenor*	deleting adj modifiers from the goal	*All Italian men want to be a great tenor*
	P2	*Some Italian men are great tenors*		*Some Italian men are great tenors*
	H	*There are Italian men who want to be a great tenor*		*There are men who want to be a great tenor*
113	P1	*Smith signed two contracts*	adding Prep. modifier to the facts	*Smith signed two contracts **In January***
	P2	*Jones also signed them*		*Jones also signed them*
	H	*Smith and Jones signed two contracts*		*Smith and Jones signed two contracts*
131	P1	*Each department has a dedicated line*	deleting Prep. modifier from the goal	*Each department has a dedicated line*
	P2	*They rent them from BT*		*They rent them from BT*
	H	*Every department rents a line **from BT***		*Every department rents a line*
002	P1	*Every Italian man wants to be a great tenor*	Using Dict. Inf-rules in the facts	*Every Italian man wants to be a **very good** tenor*
	P2	*Some Italian men are great tenors*		*Some Italian men are **very good** tenors*
	H	*There are Italian men who want to be a* `great tenor`		*There are Italian men who want to be a* `great tenor`

Table 6.8: Backtracking with approximate matching tests

Using commonsense knowledge:

Although we have a rule from the TMDC that says `(X is very good => X is great)`, it is not enough to find a proof in the last example for the problem (`fracas-002`) when we extend *'great'* to *'very good'*. Similar to the previous problem of syntactic trees, to apply one of the rules from TMDC or WDFS on the extensioned set of FraCaS problems, we have to have similar shape for the tree. Therefore, if in the last example we have in Table 6.8 the tree does not consider the term 'good' as an adjective, and 'very' is an adverb that modifies that adjective, and 'good' takes the same position that 'great' has, then we could deduce an entailment.
So, one of the factors that has an effect on our proving system is the differently structured parsed trees for similar sentences. However, we were not surprised by this issue, as some researchers

have had similar problems. For instance, when Kouylekov and Magnini [2005] tested their TexTailing system on RTE, they found 30% of the sentence pairs applied had different trees for similar expressions, which made the evaluation task for their system a non-trivial task. However, to enhance our system's attitude when dealing with such problems, there are two possible actions to take: (i) Try to use a more robust parser, but this task may be not an easy step to take because this problem is still active amongst dependency parsers; (ii) As this system can work with different types of trees, we can try something aside from dependency trees, e.g., make chunked ones. These chunked trees can be generated by adding a full stop at the end of all sentences and making all words daughters of that full stop. This action will be discussed further in any future research into this problem.

Chapter 7

Conclusion, contributions, and future work

7.1 Conclusion

In this research, we began by identifying the two well-known approaches that are used to find an entailment between given fragments of texts: *'shallow-but-robust'* and *'deep-but-brittle'*. Although the shallow approach is broadly effective, it is imprecise. Shallow approach systems can test any problem, regardless of its complex sentences, and give a result for that test. This is why it is considered a robust approach. However, many topics can confuse such systems, and that why it is considered brittle. The deep approach, however, achieves high precision, but low recall. This approach is brittle, as it lacks the ability to translate complex or long natural language sentences into logic. We then presented various systems that take in-between approaches. It is quite notable that these systems prefer to translate into logic to get two benefits: (i) To raise the percentage of the precision compared to what might be obtained through a typical shallow system, and (ii) To be able to use background knowledge (inference rules). The second reason was the key to our system here, which is to make a system that allows for a chain of steps of inference without translating the inputs to any logical format. Therefore, we have implemented a theorem prover that accepts natural language fragments and is able to find entailment using something beyond the ordinary shallow approach techniques, such as: *bag of words, string edit distance* and *tree edit distance*. This proving system contains two main algorithms: (i) The backward chaining algorithm (ii) The approximate matching algorithm.

.

- **The Backward chaining**: This is a non-deterministic algorithm because it does not know if the rule that it is going to test will lead to a solution or not. If it does not then it must backtrack to try other paths.
- **Approximate matching**: This is also a non-deterministic algorithm that contains various features. It is non-deterministic because one of its checks, skipping modifiers, is non-deterministic.

Using Horn clauses in a propositional logic inference system is possible, as that kind of logic is dealing with literals; e.g., p and q, and then there is the step of comparing them to see if they are **identical** (=) to each other or not. In FOL, the comparison is different, as the syntax of FOL is different than the propositional logic, as FOL contains formulae that are more than just literals. Therefore, the FOL-based systems use the notion of **unification:** $\oplus$. Our syntax does not match either of these categories, therefore, we changed the behaviours of the comparing operator to check for **approximate matching** ($\approx$) between the two given sentences.
The approximate matching algorithm expects inputs to be dependency trees and then compares between their parallel subtrees. It begins with trying to match the heads together;if successful, it tries to match the rest of these trees. An approximate matching can be found between two subtrees if these subtrees are equal or if one of following cases applies to the words they contain:

1. If both words w_1 and w_2 are the same (i.e., $\mathtt{w_1 == w_2}$), and they have the same monotone.
2. If *tag* (w_1) == *tag*(w_2) and w_1 is a hypernym of w_2 (i.e., $w_1 \sqsubset w_2$*), and they are in a positive context.*
3. If *tag* (w_1) == *tag*(w_2) and w_1 is a hyponym of w_2 (i.e., $w_1 \sqsupset w_2$*), and they are in a negative context.*
4. If one of the two words w_1 and w_2 is an antonym of the other, and they occur in different monotones (either upward monotone $\uparrow$ for w_1 and downward monotone for $\downarrow$ w_2, or its converse).

In addition to these rules, this system is allowed to skip adjuncts, such as prepositional modifiers and adjective modifiers. Skipping modifiers is asymmetric, like the four rules that were presented earlier. Thus, it is only allowed to drop an adjunct from a premise when the context is positive. Consequently, the system is only authorised to drop an adjunct from a hypothesis when the context is negative. The substitution relations between open-class words are taken from *WordNet* hypernyms. For substitutions amongst generalised quantifiers, this system uses a table of hierarchical relationships between generalised quantifiers collected from different sources. All of the features that are used by this system are controlled, so we can turn them on and off when we do a proof check.

In Chapter 5, we discussed the inference rules that we used in our theorem prover and how these inference rules were generated from online dictionaries, such as *TMDC* and *WDFS*. Most of the available commonsense knowledge does not suit our system. This is partially because it is not dealing with any kind of logic annotations. In addition to the rules that we extracted, there is another type of rules, syllogistic rules, that are generated from some of the double-premise FraCaS problems.

For the evaluation of this work, we evaluated the inference rules from the TMDC and WDFS dictionaries. We were able to collect more than 8,000 rules and we tested their precision by taking a human judgment approach on a sample of 200 rules, where the overall precision was 0.74. For the FraCaS test suite, a test was only done on the approximate matching algorithm. For two topics related to our research, *quantifiers* and *adjectives*, we were able to answer most of the problems that we have their right trees, with an accuracy of 0.928. The precision is considerably high, but this is because we filtered out from the parser all the problems that we might have. Then we compared the results of this matching algorithm with a baseline of a string edit distance algorithm. For two-premise problems, we tested the rules that we made from various examples. From this set of rules, we tried to extract generic rules that could cover more than one problem. For instance, if the rule for the problem (005) can solve problem (063) but not vice versa, then we delete the rule for problem (063), as it is not generic, while there is a more generic rule that can solve this problem. The last test was to try to extend the problems of FraCaS. Since the sentences in each FraCaS problem contain identical words in common, we changed these words into other forms that could be achieved only if one of our approximate matching techniques was applied.

7.2 Contributions

C1 Building an inference system that accepts inputs that are not translated into any kind of logical format (dependency trees). This system can find an entailment between two trees either using inference rules or applying TexTailment techniques. The current proof system can handle Horn clause rules, where there are one or more antecedents in the *RHS* of a rule with only one consequence in the *LHS* of that rule. The algorithm of inference that is used in this system is a backward chaining algorithm and it can be applied to multi-step inference tasks. The approximate matching algorithm can perform different techniques, such as word subsumption and deleting modifiers. Because skipping modifiers is allowed, the unification algorithm is considered to be a non-deterministic algorithm. Meaning that, this prover utilises two layers of continuations, which is not common among ordinary theorem provers. The first continuation algorithm is applied in the backtracking process, where a stack is holding the last point that the system should return in case of failing. The other continuation algorithm is applied if the unification algorithm skips a modifier. If the circumstances are encouraging it to drop a subtree, then the system will drop it and pinpoint a continuation.

C2 Creating commonsense background knowledge that fits the needs of the inference system. These rules were extracted from dictionaries, where we found definitional texts in the form 'word: definition', which are then changed into universal rules that contain variables. To generate these rules, we have turned each definitional relation into a parsable English sentence that has the conditional term 'if' in the middle. For instance, *'X is ancient if X is very old'*, to make an inference rule from this of a form $RHS \rightarrow LHS$, we replace 'if' by $\rightarrow$, and the if-clause becomes the antecedent (*LHS*), whereas the consequent (*RHS*) is the dependency tree for the main-clause part, i.e., the part that precedes 'if'. Therefore, the rule for the above example would be: `[is: VX, (?X: ???), [very: RB], [old: JJ]]` $\Rightarrow$ `[is: VX, (?X: ???), [ancient: JJ]]`. The primary source that we extracted our rules from is The Macmillan Dictionary. We then tested the validity of our rules, by applying them on WordNet definitions. Another type of rule that we extracted is some syllogistic rules that are automatically generated from FraCaS two-premise examples.

Both of the contributions were accepted in the Recent Advances in Natural Language Processing conference 2017 (RANLP 2017) and they will be published in two papers in the last quarter of 2017. The first contribution will be published under the title: *"A hybrid system to apply natural language inference over dependency trees"* and the second contribution has the following title: *"Using English dictionaries to generate commonsense knowledge in natural language"*.

7.3 Future work

Chapters 4 and 5 can be enhanced by further research. For the prover system, in its current state, this prover is able to take dependency trees and deduce an entailment between them. A problem with dealing with dependency trees is that parsing is difficult. As sentences become more complex, even the best parsers will make mistakes, and one mistake in making the parse tree—particularly one mistake high up in the tree—will lead to huge mistakes in reasoning. Instead of doing full parsing, we can parse in chunks, for instance, looking for aux+verb sequences in order to find simple NPs. That means we can do chunking instead of parsing and then turn these chunks into a tree. A tree can be made from this chunking when we make every chunk a daughter of the root node, e.g., by taking the final punctuation mark as the root, which ultimately leads to a sequence. This sequence is a sequence of chunks, rather than a sequence of words. Then we can use sequence oriented approaches, e.g., string edit distance, to work with it. We can use our extended matching algorithm to compare the elements of a pair of sequences. We could also, perhaps, use this with our inference engine to carry out N-level proofs.

For commonsense knowledge, we would like to look for more resources where we could find definitional texts that can be used as rules. A good place to seek such resources would be other dictionaries. Generally, dictionaries use short length, easy to parse definitional sentences, which make them more appropriate than any other source for this task. Some other background knowledge data employ encyclopaedia sites, such as Wikipedia. We might try to explore the nature of texts there, even though, the information in Wikipedia might not be as relevant as dictionaries because this site usually uses facts, not rules, and some of the sentences used are not easily parsed.

Bibliography

Adams, R. (2006). Textual entailment through extended lexical overlap. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, pages 128–133.

Adams, R., Nicolae, G., Nicolae, C., and Harabagiu, S. (2007). Textual entailment through extended lexical overlap and lexico-semantic matching. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 119–124. Association for Computational Linguistics.

Akhmatova, E. (2005). Textual entailment resolution via atomic propositions. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, volume 150. Citeseer.

Alabbas, M. A. S. (2013). *Textual Entailment for Modern Standard Arabic*. PhD thesis, The University of Manchester, Manchester, UK.

Allerton, D. J. (1979). *Essentials of grammatical theory: A consensus view of syntax and morphology*. Routledge.

Bachmair, L. and Ganzinger, H. (1994). Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247.

Bar-Haim, R., Dagan, I., Greental, I., and Shnarch, E. (2007). Semantic inference at the lexical-syntactic level. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 871. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Baroni, M., Bernardi, R., Do, N.-Q., and Shan, C.-c. (2012). Entailment above the word level in distributional semantics. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 23–32. Association for Computational Linguistics.

Barwise, J. and Cooper, R. (1981). Generalized quantifiers and natural language. In *Philosophy, Language, and Artificial Intelligence*, pages 241–301. Springer.

Baumgartner, P. (2000). FDPLL: A First-Order Davis-Putnam-Logeman-Loveland procedure. In *International Conference on Automated Deduction*, pages 200–219. Springer.

Beckert, B., Hähnle, R., Oel, P., and Sulzmann, M. (1996). The tableau-based theorem prover 3 t a p version 4.0. In *International Conference on Automated Deduction*, pages 303–307. Springer.

Beckert, B. and Posegga, J. (1995). leantap: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358.

Berko, J. (1958). The child's learning of english morphology. *Word*, 14(2-3):150–177.

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Black, E., Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R., and Roukos, S. (1992). Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the workshop on Speech and Natural Language*, pages 134–139. Association for Computational Linguistics.

Blackburn, P. and Bos, J. (2003). Representation and inference for natural language. *A First Course in Computational Semantics. Draft available at http://www. comsem. org, June*.

Blackburn, P. and Bos, J. (2005). Representation and inference for natural language. *A first course in computational semantics. CSLI*.

Borsley, R. (2014). *Syntactic theory: A unified approach*. Routledge.

Bos, J. (2008). Wide-coverage semantic analysis with boxer. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, pages 277–286. Association for Computational Linguistics.

Bos, J. and Markert, K. (2005). Recognising textual entailment with logical inference. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 628–635. Association for Computational Linguistics.

Bos, J. and Markert, K. (2006). When logical inference helps determining textual entailment (and when it doesnâĂŹt). In *Proceedings of the Second PASCAL RTE Challenge*, page 26.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the workshop on Speech and Natural Language*, pages 112–116. Association for Computational Linguistics.

Burridge, K. and Stebbins, T. N. (2015). *For the Love of Language: An Introduction to Linguistics*. Cambridge University Press.

Burton-Roberts, N. et al. (2016). *Analysing sentences: An introduction to English syntax*. Routledge.

Carnie, A. (2013). *Syntax: A generative introduction*. John Wiley & Sons.

Ceusters, W., Rogers, J., Consorti, F., and Rossi-Mori, A. (1999). Syntactic-semantic tagging as a mediator between linguistic representations and formal models: an exercise in linking snomed to galen. *Artificial Intelligence in Medicine*, 15(1):5–23.

Clark, P. and Harrison, P. (2008). Recognizing textual entailment with logical inference. In *Text Analysis Conference (TAC 2008) Workshop-RTE-4 Track. National Institute of Standards and Technology (NIST)*. Citeseer.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

Cooper, R. (1996). The role of situations in generalized quantifiers. *The handbook of contemporary semantic theory*, 65:86.

Cooper, R., Crouch, D., Van Eijck, J., Fox, C., Van Genabith, J., Jaspars, J., Kamp, H., Milward, D., Pinkal, M., Poesio, M., et al. (1996). Using the framework. Technical report, Technical Report LRE 62-051 D-16, The FraCaS Consortium.

Cooper, R., Crouch, R., van Eijck, J., Fox, C., Van Genabith, J., Jaspars, J., Kamp, H., Pinkal, M., Poesio, M., Pulman, S., et al. (1994). Describing the approaches. *FraCaS deliverable D*, 8:62–05.

Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

Dagan, I., Glickman, O., and Magnini, B. (2006). The pascal recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pages 177–190. Springer.

Das, G., Fleischer, R., Gasieniec, L., Gunopulos, D., and Kärkkäinen, J. (1997). Episode matching. In *Annual Symposium on Combinatorial Pattern Matching*, pages 12–27. Springer.

Dasgupta, S., Papadimitriou, C. H., and Vazirani, U. V. (2016). Algorithms.

Dinu, G. and Wang, R. (2009). Inference rules and their application to recognizing textual entailment. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 211–219. Association for Computational Linguistics.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. Cambridge, MA and London: MIT Press.

Fowler, A., Hauser, B., Hodges, D., Niles, I., Novischi, A., and Stephan, J. (2005). Applying cogex to recognize textual entailment. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, pages 69–72. Citeseer.

Ganzinger, H. and De Nivelle, H. (1999). A superposition decision procedure for the guarded fragment with equality. In *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*, pages 295–303. IEEE.

Gildea, D. and Palmer, M. (2002). The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 239–246. Association for Computational Linguistics.

Giménez, J. and Màrquez, L. (2007). Linguistic features for automatic evaluation of heterogenous mt systems. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 256–264. Association for Computational Linguistics.

Glickman, O., Dagan, I., and Koppel, M. (2005). Web based probabilistic textual entailment. In *Proceedings of the 1st Pascal Challenge Workshop*, pages 33–36.

Haegeman, L. (2009). *Thinking syntactically: a guide to argumentation and analysis*. John Wiley & Sons.

Hickl, A., Williams, J., Bensley, J., Roberts, K., Rink, B., and Shi, Y. (2006). Recognizing textual entailment with LCC‘s groundhog system. In *Proceedings of the Second PASCAL Challenges Workshop*.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). Introduction to automata theory, languages, and computation. *ACM SIGACT News*, 32(1):60–65.

Horn, A. (1951). On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21.

Icard III, T. F. (2012). Inclusion and exclusion in natural language. *Studia Logica*, 100(4):705–725.

Jaf, S. and Ramsay, A. (2015). The selection of classifiers for a data-driven parser. Libreria Editrice Cafoscarina.

Jijkoun, V., Rijke, M., et al. (2005). Recognizing textual entailment using lexical similarity.

Kayne, R. S. (2007). Several, few and many. *Lingua*, 117(5):832–858.

Kouylekov, M. and Magnini, B. (2005). Recognizing textual entailment with tree edit distance algorithms. In *Proceedings of the First Challenge Workshop Recognising Textual Entailment*, pages 17–20.

Kübler, S., McDonald, R., and Nivre, J. (2009a). Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127.

Kübler, S., McDonald, R., and Nivre, J. (2009b). Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127.

Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196.

Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707.

Levy, O., Dagan, I., Goldberger, J., and Ramat-Gan, I. (2014). Focused entailment graphs for open ie propositions. In *CoNLL*, pages 87–97.

Li, B., Wang, J. Z., Feltus, F. A., Zhou, J., and Luo, F. (2010). Effectively integrating information content and structural relationship to improve the go-based similarity measure between proteins. *arXiv preprint arXiv:1001.0958*.

Lin, D. (1998). An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304. Citeseer.

Lin, D. and Pantel, P. (2001). Dirt@ sbt@ discovery of inference rules from text. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 323–328. ACM.

Liu, H. and Singh, P. (2004). ConceptnetâĂŤa practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226.

MacCartney, B. (2009). *Natural language inference*. PhD thesis, Citeseer.

MacCartney, B. and Manning, C. D. (2007). Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 193–200. Association for Computational Linguistics.

MacCartney, B. and Manning, C. D. (2008). Modeling semantic containment and exclusion in natural language inference. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 521–528. Association for Computational Linguistics.

Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 276–283. Association for Computational Linguistics.

Magnini, B., Dagan, I., Neumann, G., and Pado, S. (2014a). Entailment graphs for text analytics in the excitement project. In *International Conference on Text, Speech, and Dialogue*, pages 11–18. Springer.

Magnini, B., Zanoli, R., Dagan, I., Eichler, K., Neumann, G., Noh, T.-G., Pado, S., Stern, A., and Levy, O. (2014b). The excitement open platform for textual inferences. In *ACL (System Demonstrations)*, pages 43–48.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). Introduction to information retrieval.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

McCune, W. (2010). Prover9 and mace4 (2005–2010). *URL http://www. cs. unm. edu/mccune/prover9*.

McCune, W. W. (1989). Otter 1. 0 users' guide. Technical report, Argonne National Lab., IL (USA).

McDonald, R. T., Nivre, J., Quirmbach-Brundage, Y., Goldberg, Y., Das, D., Ganchev, K., Hall, K. B., Petrov, S., Zhang, H., Täckström, O., et al. (2013). Universal dependency annotation for multilingual parsing. In *ACL (2)*, pages 92–97.

McGee, V. (1985). A counterexample to modus ponens. *The Journal of Philosophy*, 82(9):462–471.

Mehdad, Y. and Magnini, B. (2009). Optimizing textual entailment recognition using particle swarm optimization. In *Proceedings of the 2009 Workshop on Applied Textual Inference*, pages 36–43. Association for Computational Linguistics.

Miller, G. and Fellbaum, C. (1998). Wordnet: An electronic lexical database.

Minsky, M. (2006). The emotion machine. *New York:Simon and Schucter*.

Moldovan, D. I. and Rus, V. (2001). Logic form transformation of wordnet and its applicability to question answering. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 402–409. Association for Computational Linguistics.

Mueller, E. T. (1998). Thoughttreasure: A natural language/commonsense platform.

Napoli, D. J. (1993). Syntax: Theory and problems.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer.

Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Nivre, J., Hall, J., and Nilsson, J. (2006). Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, pages 2216–2219.

Nivre, J., Hall, J., and Nilsson, J. (2008). Memory-based dependency parsing.

Oppacher, F. and Suen, E. (1988). Harp: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4(1):69–100.

Padó, S. and Dagan, I. (2016). Textual entailment. In *Mitkov R. (Ed) The Oxford Handbook of Computational Linguistics. Second, substantially revised edition. Oxford University Press.*

Padó, S., Noh, T.-G., Stern, A., Wang, R., and Zanoli, R. (2015). Design and realization of a modular architecture for textual entailment. *Natural Language Engineering*, 21(2):167–200.

Payne, T. (2006). *Exploring language structure: a student's guide*. Cambridge University Press.

Phillips, C. (2003). Linear order and constituency. *Linguistic inquiry*, 34(1):37–90.

Poesio, M. (1994). Discourse interpretation and the scope of operators. Technical report, DTIC Document.

Radford, A. (2004). *English syntax: An introduction*. Cambridge University Press.

Raina, R., Ng, A. Y., and Manning, C. D. (2005). Robust textual inference via learning and abductive reasoning. In *AAAI*, pages 1099–1105.

Ramsay, A. (1999). Parsing with discontinuous phrases. *Natural Language Engineering*, 5(03):271–300.

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. *arXiv preprint cmp-lg/9706014*.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*.

Riazanov, A. and Voronkov, A. (1999). Vampire. In *International Conference on Automated Deduction*, pages 292–296. Springer.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41.

Robinson, J. J. (1970). Dependency structures and transformational rules. *Language*, pages 259–285.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Pearson Education.

Sankoff, D. and Kruskal, J. B. (1983). Time warps, string edits, and macromolecules: the theory and practice of sequence comparison. *Reading: Addison-Wesley Publication, 1983, edited by Sankoff, David; Kruskal, Joseph B.*, 1.

Schlicker, A., Lengauer, T., and Albrecht, M. (2010). Improving disease gene prioritization using the semantic similarity of gene ontology terms. *Bioinformatics*, 26(18):i561–i567.

Selkow, S. M. (1977). The tree-to-tree editing problem. *Information processing letters*, 6(6):184–186.

Singh, P., Lin, T., Mueller, E. T., Lim, G., Perkins, T., and Zhu, W. L. (2002). Open mind common sense: Knowledge acquisition from the general public. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1223–1237. Springer.

Slaney, J. (1993). Scott: A model-guided theorem prover. In *IJCAI*, volume 93, pages 109–114. Citeseer.

Speer, R., Havasi, C., and Lieberman, H. (2008). Analogyspace: Reducing the dimensionality of common sense knowledge. In *AAAI*, volume 8, pages 548–553.

Sriram, B., Fuhry, D., Demir, E., Ferhatosmanoglu, H., and Demirbas, M. (2010). Short text classification in twitter to improve information filtering. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 841–842. ACM.

Stickel, M., Waldinger, R., Lowry, M., Pressburger, T., and Underwood, I. (1994). Deductive composition of astronomical software from subroutine libraries. In *International Conference on Automated Deduction*, pages 341–355. Springer.

Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM.

Tammet, T. (1997). Gandalf. *Journal of Automated Reasoning*, 18(2):199–204.

Tatu, M. and Moldovan, D. (2005). A semantic approach to recognizing textual entailment. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 371–378. Association for Computational Linguistics.

Tian, R., Miyao, Y., and Matsuzaki, T. (2014). Logical inference on dependency-based compositional semantics. In *ACL (1)*, pages 79–89.

Tseng, H., Jurafsky, D., and Manning, C. (2005). Morphological features help pos tagging of unknown words across language varieties. In *Proceedings of the fourth SIGHAN workshop on Chinese language processing*, pages 32–39.

VanLehn, K. A. (1978). Determining the scope of english quantifiers. *Technical Report, Cambridge, MA: MIT.*

Vendler, Z. (1962). Each and every, any and all. *Mind*, 71(282):145–160.

Von Ahn, L., Kedia, M., and Blum, M. (2006). Verbosity: a game for collecting common-sense facts. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 75–78. ACM.

Weidenbach, C. (1999). Spass: Combining superposition, sorts and splitting. *Handbook of automated reasoning*, 2:1965–2013.

Westerståhl, D. (2011). Generalized quantifiers, e.n. zalta (ed.). *The Stanford Encyclopedia of Philosophy*.

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206.

Zang, L.-J., Cao, C., Cao, Y.-N., Wu, Y.-M., and Cun-Gen, C. (2013). A survey of commonsense knowledge acquisition. *Journal of Computer Science and Technology*, 28(4):689–719.

Appendix A

A trace of the proving algorithm

```
 1: Does the goal exist in the facts?.
 2:   unify([word(is, VX, position=1), [word(it, PR, position=0)], [word(blow,
      NN, position=3), [word(a, DT, position=2)]]] (list), [word(is, VX,
      position=1), [word(it, PR, position=0)], [word(a, DT, position=2)],
      [word(strong, JJ, position=3)], [word(effect, NN, position=4)]] (list))
 3:   (Both are lists)
 4:   Try to match current heads
 5:   unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
 6:   unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
 7:   matched, continue
 8:     unify([[word(it, PR, position=0)], [word(blow, NN, position=3),
        [word(a, DT, position=2)]]] (list), [[word(it, PR, position=0)],
        [word(a, DT, position=2)], [word(strong, JJ, position=3)],
        [word(effect, NN, position=4)]] (list))
 9:     (Both are lists)
10:     Try to match current heads
11:      unify([word(it, PR, position=0)] (list), [word(it, PR, position=0)]
        (list))
12:     (Both are lists)
13:     Try to match current heads
14:     unify(word(it, PR, position=0) (WORD), word(it, PR, position=0) (WORD))
15:     unifyWords word(it, PR, position=0) (it) word(it, PR, position=0)
        (it)
16:     matched, continue
```

```
17:       unify([[word(blow, NN, position=3), [word(a, DT, position=2)]]]
          (list), [[word(a, DT, position=2)], [word(strong, JJ, position=3)],
          [word(effect,   NN, position=4)]] (list))
18:       (Both are lists)
19:       Try to match current heads
20:       unify([word(blow, NN, position=3), [word(a, DT, position=2)]]
          (list), [word(a, DT, position=2)] (list))
21:       (Both are lists)
22:       Try to match current heads
23:         unify(word(blow, NN, position=3) (WORD), word(a, DT, position=2)
            (WORD))
24:         unifyWords word(blow, NN, position=3) (blow) word(a, DT,
            position=2) (a)
25:         Testing subsumption on words:  word(blow, NN, position=3) (n:  +),
            word(a, DT, position=2) (DT: +)
26:         Didn't unify word(blow, NN, position=3), word(a, DT, position=2)
27:       Didn't unify [word(blow, NN, position=3), [word(a, DT, position=2)]],
          [word(a, DT, position=2)]
28:       Didn't unify word(it, PR, position=0), word(it, PR, position=0)
29:     Didn't unify [word(it, PR, position=0)], [word(it, PR, position=0)]
30:   Didn't unify word(is, VX, position=1), word(is, VX, position=1)]
31: prove(goal, [facts])
32: trying this rule:  [[is:  VX, [?X: ???], [a:  DT], [strong:  JJ], [effect:
    NN]]] ==> [is:  VX, [?X: ???], [impact:  NN, [an:  DT]]]
33: (goal is:  <tb.SENTENCE instance at 0x14aad2638>)
34: unify(<tb.SENTENCE instance at 0x14aad2638> (SENTENCE), <tb.SENTENCE
    instance at 0x144ca9368> (SENTENCE))
35:   unify([word(is, VX, position=1), [word(it, PR, position=0)], [word(blow,
      NN, position=3), [word(a, DT, position=2)]]] (list), [word(is, VX,
      position=1), VAR(X, ???), [word(impact, NN, position=3), [word(a, DT,
      position=2)]]] (list))
36:   (Both are lists)
37:   Try to match current heads
38:   unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
39:     unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
40:     matched, continue
41:       unify([[word(it, PR, position=0)], [word(blow, NN, position=3),
          [word(a, DT, position=2)]]] (list), [VAR(X, ???), [word(impact, NN,
          position=3), [word(a, DT, position=2)]]] (list))
```

```
42:       (Both are lists)
43:       Try to match current heads
44:       unify([word(it, PR, position=0)] (list), VAR(X, ???)  (VARIABLE))
45:       Binding VAR(X, ???)  to [word(it, PR, position=0)]
46:       bound VAR(X, [word(it, PR, position=0)]) to [word(it, PR,
          position=0)], continue
47:         unify([[word(blow, NN, position=3), [word(a, DT, position=2)]]]
            (list), [[word(impact, NN, position=3), [word(a, DT, position=2)]]]
            (list))
48:         (Both are lists)
49:         Try to match current heads
50:         unify([word(blow, NN, position=3), [word(a, DT, position=2)]]
            (list), [word(impact, NN, position=3), [word(a, DT, position=2)]]
            (list))
51:         (Both are lists)
52:         Try to match current heads
53:         unify(word(blow, NN, position=3) (WORD), word(impact, NN,
            position=3) (WORD))
54:         unifyWords word(blow, NN, position=3) (blow) word(impact, NN,
            position=3) (impact)
55:         Testing subsumption on words:  word(blow, NN, position=3) (n:  +),
            word(impact, NN, position=3) (n:  +)
56:         Didn't unify word(blow, NN, position=3), word(impact, NN,
            position=3)
57:       Didn't unify [word(blow, NN, position=3), [word(a, DT, position=2)]],
          [word(impact, NN, position=3), [word(a, DT, position=2)]]
58:     Didn't unify [word(it, PR, position=0)], VAR(X, ???)
59:   Didn't unify word(is, VX, position=1), word(is, VX, position=1)
60: prove(goal, [facts])
61: trying this rule:  [[is:  VX, [?X: ???, ], [impact:  NN, [a:  DT]]]] ==>
    [is:  VX, [?X: ???], [blow:  NN, [a:  DT]]]
62: (goal is:  <tb.SENTENCE instance at 0x14aad2638>)
63: unify(<tb.SENTENCE instance at 0x14aad2638> (SENTENCE), <tb.SENTENCE
    instance at 0x1445cee18> (SENTENCE))
64:   unify([word(is, VX, position=1), [word(it, PR, position=0)], [word(blow,
      NN, position=3), [word(a, DT, position=2)]]] (list), [word(is, VX,
      position=1), VAR(X, ???), [word(blow, NN, position=3), [word(a, DT,
      position=2)]]] (list))
```

```
65:  (Both are lists)
66:  Try to match current heads
67:  unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
68:  unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
69:  matched, continue
70:    unify([[word(it, PR, position=0)], [word(blow, NN, position=3),
       [word(a, DT, position=2)]]] (list), [VAR(X, ???), [word(blow, NN,
       position=3), [word(a, DT, position=2)]]] (list))
71:    (Both are lists)
72:    Try to match current heads
73:    unify([word(it, PR, position=0)] (list), VAR(X, ???)  (VARIABLE))
74:    Binding VAR(X, ???)  to [word(it, PR, position=0)]
75:    bound VAR(X, [word(it, PR, position=0)]) to [word(it, PR, position=0)],
       continue
76:      unify([[word(blow, NN, position=3), [word(a, DT, position=2)]]]
         (list), [[word(blow, NN, position=3), [word(a, DT, position=2)]]]
         (list))
77:      (Both are lists)
78:      Try to match current heads
79:      unify([word(blow, NN, position=3), [word(a, DT, position=2)]] (list),
         [word(blow, NN, position=3), [word(a, DT, position=2)]] (list))
80:      (Both are lists)
81:      Try to match current heads
82:      unify(word(blow, NN, position=3) (WORD), word(blow, NN, position=3)
         (WORD))
83:      unifyWords word(blow, NN, position=3) (blow) word(blow, NN,
         position=3) (blow)
84:      matched, continue
85:        unify([[word(a, DT, position=2)]] (list), [[word(a, DT,
           position=2)]](list))
86:        (Both are lists)
87:        Try to match current heads
88:        unify([word(a, DT, position=2)] (list), [word(a, DT, position=2)]
           (list))
89:        (Both are lists)
90:        Try to match current heads
91:        unify(word(a, DT, position=2) (WORD), word(a, DT, position=2)
           (WORD))
92:        unifyWords word(a, DT, position=2) (a) word(a, DT, position=2) (a)
93:        matched, continue
```

```
94:          unify([] (list), [] (list))
95:          (Both are lists)
96:          Try to match current heads
97:          Both are empty lists
98:        Both are empty lists
99:      Both are empty lists
100:   Both are empty lists
101: Does the goal exist in the facts?
102:   unify(<tb.SENTENCE instance at 0x1445d2c20> (SENTENCE), <tb.SENTENCE
       instance at 0x14aad6ef0> (SENTENCE))
103:   unify([word(is, VX, position=1), VAR(X, [word(it, PR, position=0)]),
       [word(impact, NN, position=3), [word(a, DT, position=2)]]] (list),
       [word(is, VX, position=1), [word(it, PR, position=0)], [word(a, DT,
       position=2)], [word(strong, JJ, position=3)], [word(effect, NN,
       position=4)]] (list))
104:   (Both are lists)
105:   Try to match current heads
106:   unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
107:   unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
108:   matched, continue
109:     unify([VAR(X, [word(it, PR, position=0)]), [word(impact, NN,
         position=3), [word(a, DT, position=2)]]] (list), [[word(it,PR,
         position=0)], [word(a, DT, position=2)], [word(strong, JJ,
         position=3)], [word(effect, NN, position=4)]] (list))
110:     (Both are lists)
111:     Try to match current heads
112:     unify(VAR(X, [word(it, PR, position=0)]) (VARIABLE), [word(it, PR,
         position=0)] (list))
113:     (Both are lists)
114:     Try to match current heads
115:     unify(word(it, PR, position=0) (WORD), word(it, PR, position=0)
         (WORD))
116:     unifyWords word(it, PR, position=0) (it) word(it, PR, position=0) (it)
117:         matched, continue
118:         unify([] (list), [] (list))
119:         (Both are lists)
120:         Try to match current heads
121:         Both are empty lists
122:           unify([[word(impact, NN, position=3), [word(a, DT, position=2)]]]
               (list), [[word(a, DT, position=2)], [word(strong, JJ,
               position=3)], [word(effect, NN, position=4)]] (list))
```

```
123:          (Both are lists)
124:          Try to match current heads
125:          unify([word(impact, NN, position=3), [word(a, DT, position=2)]]
             (list), [word(a, DT, position=2)] (list))
126:          (Both are lists)
127:          Try to match current heads
128:          unify(word(impact, NN, position=3) (WORD), word(a, DT,
             position=2) (WORD))
129:          unifyWords word(impact, NN, position=3) (impact) word(a, DT,
             position=2) (a)
130:          Testing subsumption on words:  word(impact, NN, position=3) (n:
             +), word(a, DT, position=2) (DT: +)
131:          Didn't unify word(impact, NN, position=3), word(a, DT,
             position=2)
132:        Didn't unify [word(impact, NN, position=3), [word(a, DT,
           position=2)]], [word(a, DT, position=2)]
133:      Didn't unify word(it, PR, position=0), word(it, PR, position=0)
134:    Didn't unify VAR(X, [word(it, PR, position=0)]), [word(it, PR,
       position=0)]
135:  Didn't unify word(is, VX, position=1), word(is, VX, position=1)]
136: prove(goal, [facts])
137: trying this rule:  [[is:  VX, [?X: ???], [a:  DT], [strong:  JJ], [effect:
   NN]]] ==> [is:  VX, [?X: ???], [impact:  NN, [a:  DT]]]
138: (goal is:  <tb.SENTENCE instance at 0x1445d2c20>)
139: unify(<tb.SENTENCE instance at 0x1445d2c20> (SENTENCE), <tb.SENTENCE
   instance at 0x144ca9368> (SENTENCE))
140:  unify([word(is, VX, position=1), VAR(X, [word(it, PR, position=0)]),
     [word(impact, NN, position=3), [word(a, DT, position=2)]]] (list),
     [word(is, VX, position=1), VAR(X, ???), [word(impact, NN, position=3),
     [word(a, DT, position=2)]]] (list))
141:  (Both are lists)
142:  Try to match current heads
143:  unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
144:  unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
145:  matched, continue
146:    unify([VAR(X, [word(it, PR, position=0)]), [word(impact, NN,
       position=3), [word(a, DT, position=2)]]] (list), [VAR(X, ???),
       [word(impact, NN, position=3), [word(a, DT, position=2)]]] (list))
147:    (Both are lists)
148:    Try to match current heads
```

```
149:     unify(VAR(X, [word(it, PR, position=0)]) (VARIABLE), VAR(X, ???)
         (VARIABLE))
150:     Binding VAR(X, ???)  to [word(it, PR, position=0)]
151:     bound VAR(X, [word(it, PR, position=0)]) to [word(it, PR,
         position=0)], continue
152:       unify([[word(impact, NN, position=3), [word(a, DT, position=2)]]]
           (list), [[word(impact, NN, position=3), [word(a, DT, position=2)]]]
           (list))
153:       (Both are lists)
154:       Try to match current heads
155:       unify([word(impact, NN, position=3), [word(a, DT, position=2)]]
           (list), [word(impact, NN, position=3), [word(a, DT, position=2)]]
           (list))
156:       (Both are lists)
157:       Try to match current heads
158:       unify(word(impact, NN, position=3) (WORD), word(impact, NN,
           position=3) (WORD))
159:       unifyWords word(impact, NN, position=3) (impact) word(impact, NN,
           position=3) (impact)
160:       matched, continue
161:         unify([[word(a, DT, position=2)]] (list), [[word(a, DT,
             position=2)]] (list))
162:         (Both are lists)
163:         Try to match current heads
164:         unify([word(a, DT, position=2)] (list), [word(a, DT, position=2)]
             (list))
165:         (Both are lists)
166:         Try to match current heads
167:         unify(word(a, DT, position=2) (WORD), word(a, DT, position=2)
             (WORD))
168:         unifyWords word(a, DT, position=2) (a) word(a, DT, position=2) (a)
169:         matched, continue
170:           unify([] (list), [] (list))
171:         (Both are lists)
172:         Try to match current heads
173:         Both are empty lists
174:       Both are empty lists
175:     Both are empty lists
176:   Both are empty lists
177: Does the goal exist in the facts?.
178:   unify(<tb.SENTENCE instance at 0x144ca5b90> (SENTENCE), <tb.SENTENCE
       instance at 0x14aad6ef0> (SENTENCE)).
```

```
179: unify([word(is, VX, position=1), VAR(X, [word(it, PR, position=0)]),
     [word(a, DT, position=2)], [word(strong, JJ, position=3)], [word(effect,
     NN, position=4)]] (list), [word(is, VX, position=1), [word(it, PR,
     position=0)], [word(a, DT, position=2)], [word(strong, JJ, position=3)],
     [word(effect, NN, position=4)]] (list)).
180: (Both are lists)
181: Try to match current heads
182: unify(word(is, VX, position=1) (WORD), word(is, VX, position=1) (WORD))
183: unifyWords word(is, VX, position=1) (is) word(is, VX, position=1) (is)
184: matched, continue
185:   unify([VAR(X, [word(it, PR, position=0)]), [word(a, DT, position=2)],
       [word(strong, JJ, position=3)], [word(effect, NN, position=4)]] (list),
       [[word(it, PR, position=0)], [word(a, DT, position=2)], [word(strong,
       JJ, position=3)], [word(effect, NN, position=4)]] (list))
186:   (Both are lists)
187:   Try to match current heads
188:   unify(VAR(X, [word(it, PR, position=0)]) (VARIABLE), [word(it, PR,
       position=0)] (list))
189: (Both are lists)
190: Try to match current heads
191: unify(word(it, PR, position=0) (WORD), word(it, PR, position=0) (WORD))
192: unifyWords word(it, PR, position=0) (it) word(it, PR, position=0) (it)
193: matched, continue
194:   unify([] (list), [] (list))
195:   Both are empty lists
196:   unify([[word(a, DT, position=2)], [word(strong, JJ, position=3)],
       [word(effect, NN, position=4)]] (list), [[word(a, DT, position=2)],
       [word(strong, JJ, position=3)], [word(effect, NN, position=4)]] (list))
197:   (Both are lists)
198:   Try to match current heads
199:   unify([word(a, DT, position=2)] (list), [word(a, DT, position=2)]
       (list))
200:   (Both are lists)
201:   Try to match current heads
202:   unify(word(a, DT, position=2) (WORD), word(a, DT, position=2) (WORD))
203:   unifyWords word(a, DT, position=2) (a) word(a, DT, position=2) (a)
204:   matched, continue
205:     unify([] (list), [] (list))
206:     Both are empty lists
207:     unify([[word(strong, JJ, position=3)], [word(effect, NN,
         position=4)]] (list), [[word(strong, JJ, position=3)], [word(effect,
         NN, position=4)]] (list))
```

```
208:     (Both are lists)
209:     Try to match current heads
210:     unify([word(strong, JJ, position=3)] (list), [word(strong, JJ,
        position=3)] (list))
211:     (Both are lists)
212:     Try to match current heads
213:     unify(word(strong, JJ, position=3) (WORD), word(strong, JJ,
        position=3) (WORD))
214:     unifyWords word(strong, JJ, position=3) (strong) word(strong, JJ,
        position=3) (strong)
215:     matched, continue
216:       unify([] (list), [] (list))
217:       Both are empty lists
218:       unify([[word(effect, NN, position=4)]] (list), [[word(effect, NN,
          position=4)]] (list))
219:       (Both are lists)
220:       Try to match current heads
221:       unify([word(effect, NN, position=4)] (list), [word(effect, NN,
          position=4)] (list))
222:       (Both are lists)
223:       Try to match current heads
224:       unify(word(effect, NN, position=4) (WORD), word(effect, NN,
          position=4) (WORD))
225:       unifyWords word(effect, NN, position=4) (effect) word(effect, NN,
          position=4) (effect)
226:       matched, continue
227:         unify([] (list), [] (list))
228:         Both are empty lists
229:       Both are empty lists
230:     Both are empty lists
231:   Both are empty lists
232: raise exception
233: proof.SUCCESS
```

Figure A.1: A trace of the our proving algorithm

Appendix B

Randomly selected TMDC definitions

No	POS: Adjective	Pattern: "RB JJ"	No	POS: Adjective	Pattern: "RB JJ"
1	alternative:not traditional		23	disgusting:extremely unpleasant	
2	amazing:very surprising		24	dreadful:very unpleasant	
3	ancient:very old		25	dull:not intelligent	
4	angry:very annoyed		26	electric:extremely exciting	
5	asleep:not awake		27	essential:completely necessary	
6	astonishing:very surprising		28	evil:very unpleasant	
7	beautiful:very pleasant		29	excellent:extremely good	
8	boiling:extremely hot		30	explosive:very powerful	
9	brilliant:very intelligent		31	extreme:very unusual	
10	brilliant:extremely bright		32	fabulous:extremely good	
11	brutal:extremely violent		33	false:not true	
12	cheap:not expensive		34	famous:very good	
13	chronic:very bad		35	fantastic:extremely large	
14	classic:completely typical		36	freezing:very cold	
15	classic:extremely good		37	furious:extremely angry	
16	clean:not dirty		38	giant:extremely large	
17	critical:very important		39	glorious:very enjoyable	
18	crucial:extremely good		40	gorgeous:very beautiful	
19	deadly:very boring		41	heavy:very severe	
20	desirable:sexually attractive		42	horrible:very unpleasant	
21	dirty:not clean		43	immense:extremely large	
22	disastrous:very unsuccessful		44	impolite:not polite	

No	POS = Adjective	Pattern = "RB JJ"	No	POS = Adjective	Pattern = "RB JJ"
45	insufficient:not enough		67	shocking:very bad unpleasant	
46	keen:very strong		68	significant:very important	
47	key:very important		69	singular:very good	
48	little:not important		70	slim:very small	
49	lovely:very attractive		71	sober:not drunk	
50	lush:sexually attractive		72	spectacular:extremely impressive	
51	marginal:very small		73	supreme:very great	
52	massive:very severe		74	tasty:very attractive	
53	mental:mentally ill		75	tiny:extremely small	
54	minute:very small		76	ugly:very unpleasant	
55	opposite:completely different		77	unfriendly:not friendly	
56	packed:extremely crowded		78	unhappy:not satisfied	
57	partial:not complete		79	unknown:not famous	
58	particular:especially great		80	unlikely:not typical	
59	positive:completely certain		81	unreasonable:not fair	
60	powerful:physically strong		82	unreasonable:not sensible	
61	profound:very great		83	vast:extremely large	
62	profound:very severe		84	vicious:extremely violent	
63	rough:not gentle		85	wonderful:extremely good	
64	rude:not polite		86	wrong:not suitable	
65	savage:extremely severe		87	yellow:not brave	
66	separate:not related				

No	POS = Noun	Pattern = "DT NN NN"	No	POS = Noun	Pattern = "DT NN"
1	alarm:an alarm clock		23	advert:an advertisement	
2	bill:a bird’s beak		24	advocate:a lawyer	
3	boy:a male child		25	attorney:a lawyer	
4	breast:a person’s chest		26	beak:a judge	
5	breeze:a light wind		27	bill:a banknote	
6	bronze:a bronze medal		28	bloke:a man	
7	bust:a complete failure		29	bug:an insect	
8	cart:a shopping trolley		30	cable:a telegram	
9	clerk:a shop assistant		31	crisp:a crumble	
10	command:an official order		32	faith:a religion	
11	copper:a police officer		33	fall:a waterfall	
12	domain:a domain name		34	fellow:a man	
13	drink:an alcoholic drink		35	guy:a man	
14	girl:a female child		36	human:a person	
15	gum:a eucalyptus tree		37	kid:a child	
16	insult:an offensive remark		38	mate:a friend	
17	line:a telephone connection		39	motor:a car	
18	nursery:a nursery school		40	response:a reaction	
19	office:a government department		41	server:a waiter	
20	paw:a person’s hand		42	today:this day	
21	rise:an upward movement		43	tongue:a language	
22	squad:a sports team		44	utterance:a statement	

No	POS = Verb	Pattern = "TO VB NN"	No	POS = Verb	Pattern = "TO VB NN"
1	afford:to provide something		23	project:to plan something	
2	build:to develop something		24	purchase:to buy something	
3	complete:to finish something		25	raise:to build something	
4	concern:to worry someone		26	rate:to deserve something	
5	cry:to shout something		27	recall:to remember something	
6	deploy:to use something		28	reckon:to calculate something	
7	desire:to want something		29	ruin:to spoil something	
8	die:to stop operating		30	see:to understand something	
9	embody:to include something		31	signify:to mean something	
10	find:to get something		32	smoke:to produce smoke	
11	generate:to make money		33	state:to give information	
12	generate:to produce power		34	steam:to produce steam	
13	get:to kill someone		35	stream:to flow continuously	
14	grasp:to understand something		36	survey:to study something	
15	have:to do something		37	take:to need something	
16	imitate:to copy something		38	take:to remove something	
17	lick:to hit someone		39	thrust:to attack someone	
18	lift:to steal something		40	touch:to use something	
19	mark:to celebrate something		41	utilise:to use something	
20	note:to mention something		42	utter:to say something	
21	picture:to imagine something		43	vary:to change something	
22	project:to throw something		44	want:to need something	

No	POS = Adverb	Pattern = "RB"	No	POS = Adverb	Pattern = "IN DT JJ NN"
1	dead:completely		23	briefly:for a short time	
2	dead:very		24	consciously:in a deliberate way	
3	dead:directly		25	directly:in a short time	
4	directly:exactly		26	equally:to the same degree	
5	directly:immediately		27	expressly:for a particular purpose	
6	easily:definitely		28	fairly:in a fair way	
7	extraordinarily:extremely		29	halfway:to a reasonable degree	
8	fast:quickly		30	happily:in a willing way	
9	flat:completely		31	heavily:to a large degree	
10	forwards:forward		32	lately:within the recent past	
11	frequently:often		33	lightly:in a graceful way	
12	fully:completely		34	meanwhile:at the same time	
13	heavily:very		35	nicely:in a satisfactory way	
14	incredibly:extremely		36	nicely:in an attractive way	
15	instantly:immediately		37	normally:in the usual way	
16	largely:mainly		38	painfully:in a painful way	
17	likely:probably		39	politely:in a polite way	
18	nevertheless:nonetheless		40	quickly:at a fast speed	
19	poorly:badly		41	quietly:in a quiet voice	
20	precisely:exactly		42	similarly:in a similar way	
21	precisely:clearly		43	straight:in an honest way	
22	project:to throw something		44	wide:over a large area	

Appendix C

Inference rules examples

nursery (n): X is a child's room for a baby => X is a nursery
sleep (n): X is euphemisms for death => X is a sleep
mansion (n): X is a large and imposing house => X is a mansion
integrity (n): X is moral soundness => X is an integrity
swap (n): X is an equal exchange => X is a swap
rise (n): X is a growth in strength or number or importance => X is a rise
rise (n): X is a movement upward => X is a rise
politician (n): X is a person active in party politics => X is a politician
encounter (n): X is a casual or unexpected convergence => X is an encounter
school (n): X is an educational institution => X is a school
solution (n): X is a method for solving a problem => X is a solution
farmhouse (n): X is house for a farmer and family => X is a farmhouse
force (n): X is a powerful effect or influence => X is a force
horn (n): X is a brass musical instrument with a brilliant tone => X is a horn
second (n): X is a particular point in time => X is a second
study (n): X is a detailed critical inspection => X is a study
here (n): X is the present location => X is a here
protection (n): X is defense against financial failure => X is a protection
forum (n): X is a public meeting or assembly for open discussion => X is a forum
separation (n): X is the distance between things => X is a separation
substance (n): X is considerable capital => X is a substance
daughter (n): X is a female human offspring => X is a daughter
county (n): X is the largest administrative district within a state => X is a county
total (n): X is the whole amount => X is a total
strike (n): X is a conspicuous success => X is a strike
hurt (n): X is psychological suffering => X is a hurt
landscape (n): X is an extensive mental viewpoint => X is a landscape
hole (n): X is an unoccupied space => X is a hole

hold (n): X is a cell in a jail or prison => X is a hold
sleep (n): X is euphemisms for death => X is a sleep
mansion (n): X is a large and imposing house => X is a mansion
integrity (n): X is moral soundness => X is an integrity
swap (n): X is an equal exchange => X is a swap
rise (n): X is a growth in strength or number or importance => X is a rise
rise (n): X is a movement upward => X is a rise
politician (n): X is a person active in party politics => X is a politician
encounter (n): X is a casual or unexpected convergence => X is an encounter
school (n): X is an educational institution => X is a school
solution (n): X is a method for solving a problem => X is a solution
farmhouse (n): X is house for a farmer and family => X is a farmhouse
force (n): X is a powerful effect or influence => X is a force
horn (n): X is a brass musical instrument with a brilliant tone => X is a horn
second (n): X is a particular point in time => X is a second
study (n): X is a detailed critical inspection => X is a study
here (n): X is the present location => X is a here
protection (n): X is defense against financial failure => X is a protection
forum (n): X is a public meeting or assembly for open discussion => X is a forum
separation (n): X is the distance between things => X is a separation
substance (n): X is considerable capital => X is a substance
daughter (n): X is a female human offspring => X is a daughter
county (n): X is the largest administrative district within a state => X is a county
total (n): X is the whole amount => X is a total
strike (n): X is a conspicuous success => X is a strike
hurt (n): X is psychological suffering => X is a hurt
landscape (n): X is an extensive mental viewpoint => X is a landscape
hole (n): X is an unoccupied space => X is a hole
honey (n): X is a beloved person => X is a honey
must (n): X is a necessary or essential thing => X is a must
word (n): X is a brief statement => X is a word
player (n): X is an important participant => X is a player
locker (n): X is a storage compartment for clothes and valuables => X is a locker
bungalow (n): X is a small house with a single story => X is a bungalow
point (n): X is sharp end => X is a point
provincial (n): X is a country person => X is a provincial
end (n): X is a final part or section => X is an end
gate (n): X is a movable barrier in a fence or wall => X is a gate
lad (n): X is a boy or man => X is a lad
jungle (n): X is an impenetrable equatorial forest => X is a jungle
break (n): X is a personal or social separation => X is a break
bank (n): X is sloping land => X is a bank
dawn (n): X is the earliest period => X is a dawn
ring (n): X is a characteristic sound => X is a ring
ring (n): X is a toroidal shape => X is a ring
reader (n): X is a literate person => X is a reader
surprise (n): X is a sudden unexpected event => X is a surprise
turning (n): X is a movement in a new direction => X is a turning
resume (n): X is short descriptive summary => X is a resume
formation (n): X is a particular spatial arrangement => X is a formation
struggle (n): X is strenuous effort => X is a struggle
kingdom (n): X is the domain ruled by a king or queen => X is a kingdom
cereal (n): X is rice => X is a cereal
cereal (n): X is oats => X is a cereal
distance (n): X is a distant region => X is a distance
distance (n): X is a remote point in time => X is a distance
palace (n): X is a large ornate exhibition hall => X is a palace
mind (n): X is an important intellectual => X is a mind
principle (n): X is a basic truth or law or assumption => X is a principle
hunger (n): X is strong desire for something => X is a hunger
queen (n): X is a female sovereign ruler => X is a queen

radio (n): X is medium for communication => X is a radio
bust (n): X is a complete failure => X is a bust
hostility (n): X is a hostile disposition => X is a hostility
watch (n): X is a small portable timepiece => X is a watch
leisure (n): X is time available for ease and relaxation => X is a leisure
coast (n): X is the area within view => X is a coast
pocket (n): X is an enclosed space => X is a pocket
wedding (n): X is the nuptial ceremony => X is a wedding
report (n): X is an essay => X is a report
mud (n): X is soft wet earth => X is a mud
heating (n): X is a rising temperature => X is a heating
handful (n): X is a small number or amount => X is a handful
relaxation (n): X is freedom from activity => X is a relaxation
height (n): X is the highest level or degree attainable => X is a height
blonde (n): X is a person with fair skin and hair => X is a blonde
life (n): X is the period between birth and the present time => X is a life
life (n): X is a living person => X is a life
lift (n): X is a ride in a car => X is a lift
need (n): X is the reason for the action => X is a need
tissue (n): X is a soft thin paper => X is a tissue
independent (n): X is a neutral or uncommitted person => X is an independent
hand (n): X is physical assistance => X is a hand
cherry (n): X is a red fruit with a single hard stone => X is a cherry
ease (n): X is freedom from activity => X is an ease
retreat (n): X is a place affording peace and quiet => X is a retreat
specific (n): X is a fact about some part => X is a specific
security (n): X is financial independence => X is a security
old (n): X is past times => X is an old
dear (n): X is a beloved person => X is a dear
fox (n): X is a shifty deceptive person => X is a fox
head (n): X is an individual person => X is a head
wireless (n): X is medium for communication => X is a wireless
daddy (n): X is an informal term for a father => X is a daddy
flood (n): X is an overwhelming number or amount => X is a flood
welcome (n): X is a greeting or reception => X is a welcome
bullet (n): X is a high-speed passenger train => X is a bullet
time (n): X is an indefinite period => X is a time
time (n): X is a suitable moment => X is a time
time (n): X is a person’s experience on a particular occasion => X is a time
charge (n): X is financial liabilities => X is a charge
sovereignty (n): X is government free from external control => X is a sovereignty
division (n): X is an administrative unit in government or business => X is a division
minute (n): X is a particular point in time => X is a minute
minute (n): X is a short note => X is a minute
level (n): X is height above ground => X is a level
trend (n): X is the popular taste at a given time => X is a trend
boost (n): X is an increase in cost => X is a boost
witch (n): X is a female sorcerer or magician => X is a witch
box (n): X is a container => X is a box
box (n): X is the driver’s seat on a coach => X is a box
box (n): X is separate partitioned area in a public place for a few people => X is a box
boy (n): X is a male human offspring => X is a boy
love (n): X is a beloved person => X is a love
extra (n): X is a minor actor in crowd scenes => X is an extra
jar (n): X is a sudden jarring impact => X is a jar
effort (n): X is hard work => X is an effort
effort (n): X is a notable achievement => X is an effort
behalf (n): X is for someone’s benefit => X is a behalf
car (n): X is where passengers ride up and down => X is a car
soul (n): X is deep feeling or emotion => X is a soul

represent (v): X describe Y in a particular way, especially when this influences other people's opinions => X represent
code (v): X mark Y with a code that gives information about it => X code
consider (v): X think about Y carefully before making a decision or developing an opinion => X consider
consider (v): X think that Y may exist or may be true => X consider
consider (v): X look at Y in a particular way => X consider
skip (v): X avoid doing or having Y => X skip
broaden (v): X make Y include more things or people => X broaden
relieve (v): X replace Y when they finish work => X relieve
soften (v): X become softer, or to make Y softer => X soften
soften (v): X become kinder and less severe, or to make Y do this => X soften
soften (v): X make Y look more pleasant by making its colour or shape less strong => X soften
protest (v): X try to make other people believe that Y is true => X protest
coach (v): X teach Y a special skill, especially one connected with performing in public => X coach
coach (v): X tell Y what to say or do in a particular situation => X coach
block (v): X stop Y from moving through or along something else => X block
block (v): X use your power to stop Y from being done or from succeeding => X block
follow (v): X watch where Y is going => X follow
follow (v): X understand something, especially Y long or complicated => X follow
hate (v): X dislike Y very much => X hate
send (v): X post a letter or parcel to Y => X send
send (v): X make Y move or fall suddenly => X send
send (v): X make Y feel a particular emotion => X send
send (v): X allow a substance such as smoke or Y made by a chemical process to escape into the atmosphere => X send
amaze (v): X surprise Y very much, especially by being very impressive => X amaze
charge (v): X ask Y to pay an amount of money for something that you are selling to them or doing for them => X charge
charge (v): X attack Y by running very fast towards them => X charge
charge (v): X make Y officially responsible for doing something => X charge
compensate (v): X change or remove the bad result of Y => X compensate
compensate (v): X pay Y money because they have suffered an injury or loss => X compensate
breach (v): X get through Y such as a wall or fence => X breach
volunteer (v): X offer or choose to do Y without being forced => X volunteer
reveal (v): X let Y become known, for example a secret or information that was previously not known => X reveal
reveal (v): X show Y that was covered or hidden => X reveal
torture (v): X make Y feel extremely worried or upset about something => X torture
distort (v): X change Y such as information so that it is no longer true or accurate => X distort
embark (v): X get on a ship in order to begin a journey, or to put Y on a ship => X embark
decide (v): X consider Y carefully and officially state what should be done about it => X decide
boot (v): X kick Y hard => X boot
trouble (v): X make Y worried => X trouble
blast (v): X hit Y with a lot of energy or force => X blast
wipe (v): X clean or dry Y by moving a cloth or something soft over it => X wipe
encounter (v): X meet Y or to see something for the first time => X encounter
arrange (v): X make plans for Y to happen, for example by agreeing a time and place => X arrange
disturb (v): X interrupt Y and stop them from continuing what they were doing => X disturb
disturb (v): X make Y move => X disturb
disturb (v): X do Y that stops a place or situation from being pleasant, calm, or peaceful => X disturb
exaggerate (v): X describe Y in a way that makes it seem better, worse, larger, more important etc than it really is => X exaggerate
exaggerate (v): X make Y seem more extreme => X exaggerate
shock (v): X make Y feel embarrassed or offended by saying or doing something offensive or immoral => X shock
shock (v): X give Y an electric shock => X shock
settle (v): X decide Y definitely => X settle
settle (v): X put Y carefully in a place => X settle

settle (v): X begin to have an effect on Y => X settle
round (v): X go round Y => X round
round (v): X make Y round or curved => X round
shave (v): X pass very close to Y => X shave
prevent (v): X stop Y from happening => X prevent
prevent (v): X stop Y from doing something => X prevent
pretend (v): X imagine that Y is true when you are playing a game => X pretend
pretend (v): X claim that Y is true when it is not => X pretend
force (v): X make Y do something that they do not want to do, for example by using or threatening to use violence => X force
force (v): X use physical force to move Y in a particular direction => X force
force (v): X make Y happen => X force
hack (v): X cut Y in a rough way, with a lot of energy, or many times => X hack
discover (v): X find Y that was missing or hidden => X discover
discover (v): X recognise the ability of Y such as a writer or performer and help to make them famous => X discover
nail (v): X prove that Y has done something wrong or illegal => X nail
nail (v): X do Y in a perfect way, especially in sport => X nail
imply (v): X suggest that you think Y without saying it directly => X imply
cost (v): X cause Y to lose something good or valuable => X cost
cost (v): X calculate exactly how much Y will cost => X cost
design (v): X decide how Y will be made, including how it will work and what it will look like, and often to make drawings of it => X design
plead (v): X ask for Y in an urgent or emotional way => X plead
plead (v): X mention Y as an excuse for doing or not doing something => X plead
plead (v): X try to show that Y is important or worth trying to achieve => X plead
curse (v): X use magic powers to make bad things happen to Y => X curse
hide (v): X put Y in a place so that no one can find or see it => X hide
hide (v): X make Y difficult or impossible to see clearly => X hide
squeeze (v): X make Y have financial trouble, for example by raising prices, cutting a supply of money, or increasing competition => X squeeze
favour (v): X help Y and give them an advantage in an unfair way => X favour
favour (v): X make a situation easier or better for Y => X favour
melt (v): X make Y kinder and more sympathetic => X melt
crush (v): X hit or press Y so hard that you damage it severely or destroy it, especially by making its shape flatter => X crush
crush (v): X make Y feel disappointed, embarrassed, or upset => X crush
experiment (v): X perform scientific tests in order to find out what happens to Y in particular conditions => X experiment
endorse (v): X express support for Y, especially in public => X endorse
nurse (v): X look after Y who is ill or injured => X nurse
inspect (v): X look at Y carefully in order to check that it is correct or good enough => X inspect
install (v): X put Y somewhere => X install
exchange (v): X give Y something in return for something that they give you => X exchange
exchange (v): X say Y to someone and then listen to what they say => X exchange
modify (v): X change Y slightly, especially in order to improve it or to make it less extreme => X modify
blackmail (v): X make Y give you money or do what you want by threatening to tell people embarrassing information about them => X blackmail
contemplate (v): X consider doing Y in the future => X contemplate
contemplate (v): X think very carefully about Y for a long time => X contemplate
contemplate (v): X look at Y for a long time => X contemplate
explore (v): X travel to a place in order to learn about it or to search for Y valuable such as oil => X explore
let (v): X allow Y to happen => X let
let (v): X rent a room, flat, house etc to Y => X let
free (v): X let Y leave a prison or a place where they have been forced to stay => X free
free (v): X help Y to get out of a place => X free
free (v): X remove Y unpleasant that affects someone or limits their behaviour => X free

foul (a): X is offensively malodorous => X is foul
known (a): X is apprehended with certainty => X is known
flash (a): X is tastelessly showy => X is flash
young (a): X is before complete maturity => X is young
literary (a): X is knowledgeable about literature => X is literary
stable (a): X is subject to little fluctuation => X is stable
choice (a): X is appealing to refined taste => X is choice
gloomy (a): X is depressingly dark => X is gloomy
fragile (a): X is a fragile claim to fame" => X is fragile
large (a): X is ostentatiously lofty in style => X is large
small (a): X is faint => X is small
pragmatic (a): X is concerned with practical matters => X is pragmatic
helpless (a): X is without help => X is helpless
new (a): X is before complete maturity => X is new
new (a): X is unfamiliar => X is new
active (a): X is in operation => X is active
great (a): X is uppercase => X is great
involved (a): X is enveloped => X is involved
fantastic (a): X is foolish => X is fantastic
fantastic (a): X is extravagantly fanciful in design, construction, appearance => X is fantastic
secure (a): X is easy in mind => X is secure
visible (a): X is obvious to the eye => X is visible
working (a): X is adequate for practical use => X is working
hungry (a): X is feeling hunger => X is hungry
live (a): X is possessing life => X is live
live (a): X is rebounds readily => X is live
mysterious (a): X is beyond ordinary understanding => X is mysterious
capital (a): X is uppercase => X is capital
more (a): X is quantifier meaning greater in number => X is more
flat (a): X is horizontally level => X is flat
acceptable (a): X is meeting requirements => X is acceptable
acceptable (a): X is adequate for the purpose => X is acceptable
warm (a): X is psychologically warm => X is warm
proof (a): X is able to withstand => X is proof
crazy (a): X is foolish => X is crazy
prompt (a): X is on time => X is prompt
slim (a): X is small in quantity => X is slim
serial (a): X is in regular succession without gaps => X is serial
dress (a): X is suitable for formal occasions => X is dress
hot (a): X is extended meanings => X is hot
pure (a): X is without qualification => X is pure
pure (a): X is free from discordant qualities => X is pure
pure (a): X is sinless => X is pure
intended (a): X is future => X is intended
intended (a): X is betrothed => X is intended
southern (a): X is from the south => X is southern
mad (a): X is very foolish => X is mad
arch (a): X is expert in skulduggery => X is arch
harsh (a): X is disagreeable to the senses => X is harsh
short (a): X is low in stature => X is short
natural (a): X is in accordance with nature => X is natural
natural (a): X is unthinking => X is natural
natural (a): X is free from artificiality => X is natural
effective (a): X is ready for service => X is effective
light (a): X is psychologically light => X is light
light (a): X is nimble => X is light
tall (a): X is high in stature => X is tall
tall (a): X is lofty in style => X is tall
typical (a): X is conforming to a type => X is typical
better (a): X is more than half => X is better: 0

good (a): X is morally admirable => X is good
good (a): X is appealing to the mind => X is good
good (a): X is in excellent physical condition => X is good
good (a): X is beneficial to health => X is good
adjacent (a): X is abutting => X is adjacent
stout (a): X is euphemisms for ‘fat’ => X is stout
presidential (a): X is befitting a president => X is presidential
stock (a): X is routine => X is stock
restless (a): X is ceaselessly in motion => X is restless
anxious (a): X is eagerly desirous => X is anxious
unnatural (a): X is contrary to nature => X is unnatural
correct (a): X is free from error => X is correct
pathetic (a): X is inspiring scornful pity => X is pathetic
sideways (a): X is at an angle => X is sideways
little (a): X is faint => X is little
little (a): X is low in stature => X is little
pleasant (a): X is affording pleasure => X is pleasant
base (a): X is debased => X is base
punk (a): X is flimsy => X is punk
mortal (a): X is subject to death => X is mortal
loud (a): X is tastelessly showy => X is loud
philosophical (a): X is meeting trouble with level-headed detachment => X is philosophical
fast (a): X is conducive to rapid speeds => X is fast
thick (a): X is abounding => X is thick
open (a): X is ready for business => X is open
white (a): X is unsullied => X is white
white (a): X is without malicious intent => X is white
inner (a): X is exclusive to a centre => X is inner
fluid (a): X is variable => X is fluid
fluid (a): X is affording change => X is fluid
ridiculous (a): X is inspiring scornful pity => X is ridiculous
ridiculous (a): X is incongruous => X is ridiculous
ridiculous (a): X is resembling farce => X is ridiculous
wide (a): X is great in degree => X is wide
kind (a): X is agreeable, conducive to comfort => X is kind
silly (a): X is ludicrous, foolish => X is silly
silly (a): X is inspiring scornful pity => X is silly
cleaner (a): X is free from impurities => X is cleaner
cleaner (a): X is free from objectionable elements => X is cleaner
cleaner (a): X is free from clumsiness => X is cleaner
gradual (a): X is proceeding in small stages => X is gradual
better (a): X is appealing to the mind => X is better
better (a): X is in excellent physical condition => X is better
better (a): X is beneficial to health => X is better
false (a): X is inaccurate in pitch => X is false
false (a): X is unfaithful => X is false
outstanding (a): X is distinguished from others in excellence => X is outstanding
ideal (a): X is embodying an ideal => X is ideal
potential (a): X is in prospect => X is potential
interior (a): X is inside the country => X is interior
successive (a): X is in regular succession without gaps => X is successive
mobile (a): X is affording change => X is mobile
clear (a): X is readily apparent to the mind => X is clear
unreasonable (a): X is beyond normal limits => X is unreasonable
clean (a): X is free from impurities => X is clean
clean (a): X is free from objectionable elements => X is clean
express (a): X is without unnecessary stops => X is express
endless (a): X is infinitely great in number => X is endless
clean (a): X is free from clumsiness => X is clean

directly (r): X exactly Y => X directly
directly (r): X immediately Y => X directly Y
forwards (r): X forward Y => X forwards Y
flat (r): X completely Y => X flat Y
soon (r): X quickly Y => X soon Y
dead (r): X completely Y => X dead Y
dead (r): X very Y => X dead Y
dead (r): X directly Y => X dead Y
likely (r): X probably Y => X likely Y
primarily (r): X mainly Y => X primarily Y
precisely (r): X exactly Y => X precisely Y
precisely (r): X clearly Y => X precisely Y
readily (r): X easily Y => X readily Y
fast (r): X quickly Y => X fast Y
seriously (r): X very Y => X seriously Y
low (r): X quietly Y => X low Y
poorly (r): X badly Y => X poorly Y
instantly (r): X immediately Y => X instantly Y
regularly (r): X frequently Y => X regularly Y
heavily (r): X very Y => X heavily Y
truly (r): X very Y => X truly Y
quick (r): X quickly Y => X quick Y
vaguely (r): X slightly Y => X vaguely Y
frequently (r): X often Y => X frequently Y
promptly (r): X immediately Y => X promptly Y
totally (r): X completely Y => X totally Y
wholly (r): X completely Y => X wholly Y
easily (r): X definitely Y => X easily Y
incredibly (r): X extremely Y => X incredibly Y
largely (r): X mainly Y => X largely Y
extraordinarily (r): X extremely Y => X extraordinarily Y
nevertheless (r): X nonetheless Y => X nevertheless Y
loud (r): X loudly Y => X loud Y
fully (r): X completely Y => X fully Y
principally (r): X mainly Y => X principally Y
typically (r): X usually Y => X typically Y
ill (r): X badly Y => X ill Y
consciously (r): X does Y in a deliberate way => X does Y consciously
quietly (r): X does Y in a quiet voice => X does Y quietly
rightly (r): X does Y for a good reason => X does Y rightly
directly (r): X does Y in a short time => X does Y directly
momentarily (r): X does Y for a moment => X does Y momentarily
politely (r): X does Y in a polite way => X does Y politely
together (r): X does Y at the same time => X does Y together
halfway (r): X does Y to a reasonable degree => X does Y halfway
tomorrow (r): X does Y on the day after today => X does Y tomorrow
tomorrow (r): X does Y in the future => X does Y tomorrow
strangely (r): X does Y in an unusual way => X does Y strangely
lightly (r): X does Y in a graceful way => X does Y lightly
slow (r): X does Y at a slow speed => X does Y slow
ideally (r): X does Y in the best possible way => X does Y ideally
upwards (r): X does Y towards a higher position => X does Y upwards
similarly (r): X does Y in a similar way => X does Y similarly
strictly (r): X does Y in a strict way => X does Y strictly
currently (r): X does Y at the present time => X does Y currently
lately (r): X does Y within the recent past => X does Y lately
quickly (r): X does Y at a fast speed => X does Y quickly
wide (r): X does Y over a large area => X does Y wide
wildly (r): X does Y in an uncontrolled way => X does Y wildly
equally (r): X does Y to the same degree => X does Y equally
fairly (r): X does Y in a fair way => X does Y fairly
briefly (r): X does Y for a short time => X does Y briefly

warmly (r): X does Y in a friendly way => X does Y warmly
warmly (r): X does Y with great enthusiasm => X does Y warmly
consequently (r): X does Y as a result => X does Y consequently
meanwhile (r): X does Y at the same time => X does Y meanwhile
happily (r): X does Y in a happy way => X does Y happily
happily (r): X does Y in a willing way => X does Y happily
heavily (r): X does Y to a large degree => X does Y heavily
heavily (r): X does Y in large amounts => X does Y heavily
yesterday (r): X does Y on the day before today => X does Y yesterday
surely (r): X does Y without any doubt => X does Y surely
sharply (r): X does Y in a severe way => X does Y sharply
painfully (r): X does Y in a painful way => X does Y painfully
kindly (r): X does Y in a kind way => X does Y kindly
nicely (r): X does Y in a satisfactory way => X does Y nicely
nicely (r): X does Y in an attractive way => X does Y nicely
always (r): X does Y on every occasion => X does Y always
originally (r): X does Y at first => X does Y originally
expressly (r): X does Y for a particular purpose => X does Y expressly
normally (r): X does Y in the usual way => X does Y normally
formerly (r): X does Y in the past => X does Y formerly
definitely (r): X does Y without any doubt => X does Y definitely
politically (r): X does Y in a political way => X does Y politically
secondly (r): X does Y in the second place => X does Y secondly
politely (r): X does Y in a polite manner => X does Y politely
heavily (r): X does Y to a considerable degree => X does Y heavily
heavily (r): X does Y in a heavy-footed manner => X does Y heavily
heavily (r): X does Y with great force => X does Y heavily
indeed (r): X does Y in truth => X does Y indeed
comparatively (r): X does Y in a relative manner => X does Y comparatively
soon (r): X does Y in the near future => X does Y soon
bright (r): X does Y with brightness => X does Y bright
ever (r): X does Y at any time => X does Y ever
ever (r): X does Y at all times => X does Y ever
through (r): X does Y over the whole distance => X does Y through
through (r): X does Y to completion => X does Y through
through (r): X does Y throughout the entire extent => X does Y through
remarkably (r): X does Y in a signal manner => X does Y remarkably
fine (r): X does Y in a delicate manner => X does Y fine
slow (r): X does Y without speed => X does Y slow
formerly (r): X does Y at a previous time => X does Y formerly
tight (r): X does Y in an attentive manner => X does Y tight
behind (r): X does Y in debt => X does Y behind
actually (r): X does Y in actual fact => X does Y actually
actually (r): X does Y at the present moment => X does Y actually
meantime (r): X does Y during the intervening time => X does Y meantime
only (r): X does Y in the final outcome => X does Y only
deliberately (r): X does Y with intention => X does Y deliberately
deliberately (r): X does Y in an intentional manner => X does Y deliberately
deliberately (r): X does Y in a deliberate unhurried manner => X does Y deliberately
hopefully (r): X does Y with hope => X does Y hopefully
hopefully (r): X does Y in a hopeful manner => X does Y hopefully
second (r): X does Y in the second place => X does Y second
asleep (r): X does Y into a sleeping state => X does Y asleep
flush (r): X does Y in the same plane => X does Y flush
substantially (r): X does Y in a strong substantial way => X does Y substantially
close (r): X does Y in an attentive manner => X does Y close
beyond (r): X does Y in addition => X does Y beyond
outward (r): X does Y toward the outside => X does Y outward
really (r): X does Y in actual fact => X does Y really
really (r): X does Y in fact => X does Y really
fundamentally (r): X does Y in essence => X does Y fundamentally