NATURAL LOGIC AND NATURAL DEDUCTION FOR REASONING ABOUT NATURAL LANGUAGE

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE SCHOOL OF ENGINEERING

2021

By
Ghadah Binhadba
Department of Computer Science

Contents

List of Tables

List of Figures

Abstract

NATURAL LOGIC AND NATURAL DEDUCTION FOR REASONING ABOUT NATURAL LANGUAGE

Ghadah Binhadba

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2021

One way of measuring a natural language processing system's semantic capacity is by demonstrating that it can handle natural language inference (NLI). Automating the task of NLI requires natural language text to be converted into a meaning representation (MR) on which inference procedures can be applied. Finding a balance between the expressive power of a MR and the inferential capabilities that could be conducted using such representation is a challenge. In that respect, the aim of this thesis is to reason about deep semantic phenomena (such as defaults, monotonicity, quantifiers and propositional attitudes) and thus construct a MR that is able to preserve the subtle semantic distinctions people make when reasoning about such phenomena. For decades, the best way for obtaining such semantically deep MRs was by translating NL texts into some formal language such as first-order logical formulas. However, such translations have proven difficult. Alternatively, and motivated by the success of natural logic systems on the pairwise entailments tasks, in which MRs are close to NL texts' surface form, we have investigated the use of a first-order logic theorem prover (to allow reasoning over multi-premises tasks) and have adapted it to work on representations that were built based on syntactical analysis (dependency trees) of NL texts. To ensure the right depth of representation, we have investigated the literature to learn about the intended semantic phenomena and hence model them in our MR accordingly. To measure such an approach's performance, we have implemented an inference system

that consists of three parts: a dependency grammar to generate syntactical trees, a tree normalizer to build the desired MRs (namely inference friendly forms) and a theorem prover (CSATCHMO+) along with some necessary background information. To our knowledge, there is not much work that has been done in this line of research (combining theorem proving with a version of natural logic). Therefore, we have tested our system's performance on the part of a test-set that has a clear representation of semantic phenomena (The FraCas test-set) and compared it to the literature's related systems. Overall, the current findings encourage further investigation extending to other phenomena, as we obtained a comparable result when compared to the related natural logic systems and an only just better result than a natural logic theorem prover.

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

Dedication

This work is dedicated to my beloved parents, my late father *Rafed* and my mother *Latifah*, whose love and devotion, endless support and prayers, have been the strength of my striving in all aspects of my life especially this journey.

Acknowledgements

First and foremost, I want to thank my supervisor, *Professor Allan Ramsay*. It has been an honour to carry out this research under his supervision. I appreciate every aspect of his contributions, in terms of time and ideas, to ensure my PhD experience was productive and rewarding.

I would like also to thank my sponsor, *King Saud University*, for the huge opportunity I have been granted to continue my study in one of the worlds leading universities.

My most profound gratitude goes to: my life-partner, *Mohammed Almutairi* who I would not have been able to achieve this without his love and support; my sweet daughter *Toleen*, who has blessed me with a life full of joy and happiness; my sisters, *Sarah* (thank you for being a second mum for Toleen especially during the pandemic year) *Maryam*, *Nada*, and *Albandry* (thank you for being amazing aunties as well and most importantly thank you for letting me vent every night about every step of creating this thesis); my brothers, *Ahmed* and *Yazeed* (thank you for accompanying me and giving my dream a priority) our eldest *Saad* and *Mishal* (thank you for your support, words of wisdom, and teasing – guess what! I made it!) *Mohammed* (thank you for being Toleen's best friend and keeping her occupied during quarantine).

Last but not least, I would like to extend my gratitude to all of my friends and colleagues at the University of Manchester for their gracious help and consistent support.

List of Abbreviations

AP	Adjective Phrase
BNP	Bare Noun Phrase
BP	Bare Plural
Det	Determiner
DNP	Definite Noun Phrase
DS	Data Structure
DT	Dependency Tree
ES	Existential Sentence
FOL	First-Order Logic
GQ	Generalized Quantifier
HOL	High-Order Logic
IFF	Inference Friendly Form
IR	Information Retrieval
LHS	Left-hand side
NL	Natural Language
NLU	Natural Language Understanding
NLI	Natural Language Inference
NP	Noun Phrase
MR	Meaning Representation
PP	Prepositional Phrase
PDT	Processed Dependency Tree
QA	Question Answering
QNP	Quantified Noun Phrase
QLF	Quasi Logical Form
RC	Relative Clause
RE	Referring Expression
RQLF	Resolved Quasi Logical Form
RHS	Right-hand side
SF	Skolem Function
SubC	Subordinate Clause
VP	Verb Phrase

Chapter 1

Introduction

For people, understanding a *Natural Language* (NL) text amounts to understanding the information conveyed by that text, updating their knowledge when learning something new, and hence drawing the relevant consequences [Iwańska, 1993; Chatzikyriakidis et al., 2017]. For instance, upon reading the sentences '*Fido is a dog.*' and '*All animals are mortals.*', a person may reasonably infer (from these two sentences and from the prior-knowledge that '*a dog*' is '*an animal*') that '*Fido is mortal*'. This pattern of human reasoning is called *inference*[1]. In the field of *natural language processing* (NLP), determining whether a NL text can be expressed by, or inferred from another, as would reasonably be done by a human, can be leveraged by many NLP applications (e.g. *question answering*(QA), *information retrieval*(IR), *summarization*, and *machine translation*), and is called *natural language inference* (NLI) [Glickman et al., 2005; MacCartney, 2009].

Automating the process of NLI is the main focus of this thesis. It requires NL texts to be transformed into some sort of *meaning representations* (MRs), and to use an *inference engine* that can reason with these representations (see Figure 1.1).

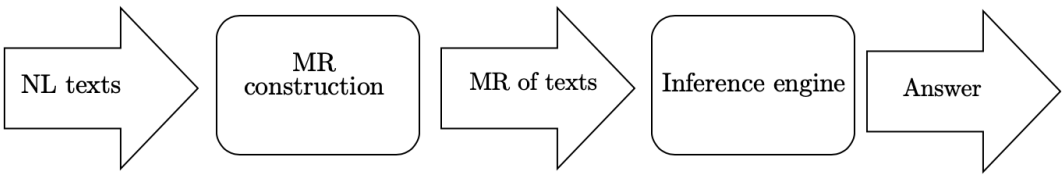


Figure 1.1: The general architecture of a computational inference system.

[1] "The notion of inference the current NLP systems are learning, is much narrower compared to the range of inference patterns found in human reasoning."[Bernardy and Chatzikyriakidis, 2019]

As [Bos, 2008] argues, a challenge one might face is not only how to adequately represent meaning, but also how to compromise between the expressive power of a MR and the practical reasoning capabilities that the inference engine can conduct using this representation.

In that respect, there exists a spectrum of approaches to the problem of NLI. At one end of the spectrum lie **robust-but-shallow** approaches where inferences are computed based on some lexical similarities between NL texts' surface forms or the bag of their words. For example, consider the NL *argument*[2] in (1.1):

(1.1)
p: Several men love Mary.
---
c: Some men like Mary.

A *bag-of-words* model like that of Glickman et al. [2005] or Jijkoun et al. [2005] will conclude that a consequence c can be inferred from a premise p and hence the argument is valid, if every word in c is matched to the most lexically similar (based on some measures) word in p; beside identical words, '*some*' is matched to '*several*' and '*like*' to '*love*'. A shallow approach like this, although broadly effective, is terribly imprecise [MacCartney, 2009]. In particular there is no consideration of predicate-argument structure and hence such an approach considers '*Mary likes some men*' a valid consequence from p in (1.1), even though the roles of those involved in the **like** event have been reversed. Such drawbacks can be amended by considering syntactical information (e.g. [MacCartney, 2009]) or surface forms and alignment method (e.g [Adams, 2006]). Nonetheless, shallow approaches, in general, will still stumble over common phenomena such as antonyms and negation. More importantly, they tend to ignore an important semantic feature, namely *monotonicity*[3].

At the other end of the spectrum there are **deep-but-brittle** approaches. Given a NL argument, approaches of this kind rely on performing a full semantic analysis of both of the argument's constituents (p and c) and then applying *inference rules* on these analyses to construct a chain of formal reasoning steps from p to c. For logicians and semanticists, the most obvious way to achieve the desired depth of analysis of meaning is to translate p and c into formulas of some formal language such as *first-order logic*

[2] A NL argument consists of several sentences, of which one is the consequence and the others are premises [Blackburn and Bos, 2005].

[3] Monotonicity is considered a semantic feature of lexical items and constructions that affect the direction of inference; in an upward-monotone context one can infer from a specific expression a more general one (e.g. '*John is walking*' $\models$ '*John is moving*'). In contrast the downward-monotone context allows inferring a specific expression from more general one (e.g. '*John did not move*' $\models$ '*John did not walk*'). A complete illustration of monotonicity is given in Chapter 4.

(FOL) and then use those translations in a proof search method (such as a theorem prover or a model builder) to create a formal proof of c from p.

FOL, among other kinds of logic, has been favoured for decades due to its expressiveness and inferential effectiveness [Bos, 2011]. Computationally, FOL proof systems have made enormous progress through a number of *automatic theorem provers* and *model builders* (both discussed in Chapter 3). Nonetheless, in contrast to shallow approaches, FOL-based approaches fail in open-domain settings as NL texts tend to be fairly complex and hence systems trying to construct their FOL equivalences are faced with countless thorny problems such ambiguity, ellipsis, idioms, and so on.

Beside the difficulty of translating into logic, FOL fails to capture the meaning of important semantic phenomena such as *default* sentences, e.g. '*Most birds fly*' and '*swans are probably white*'. One reason is that, in contrast to FOL, reasoning about defaults is *non-monotonic* [Antoniou, 1999], which means that a conclusion drawn from a default sentence might be changed or refuted when more information is provided.

Another phenomenon that systems based on FOL are unable to handle is propositional attitudes – verbs such as *believe* and *know*. Statements of such attitude terms have been treated in FOL as modalities. However, [Barr, 1980] explained that this treatment could lead to mistaken inference. For example, treating knowledge as a modality would comply with the following rule: $k\alpha \wedge k(\alpha \rightarrow \beta) \vdash k\beta$, which means that if you know α and you know that β follows from α, then you know β) – not that you **could find out** β, but that you **already know** β). Obviously, this is not true because knowing something does not necessarily mean knowing everything that could follow from it.

Higher-order logic (HOL), such as *Montague* semantics, is a language with more expressive power than FOL which does not have a problem creating a formal representations for the above phenomena. Nonetheless, to date, there is no efficient theorem prover for the language of HOL that we are aware of.

Natural logic is a logic that has revived in the last two decades as a **mid-ground approach** to the problem of NLI. It generally refers to the regularities that can prove a valid NL argument by operating on representations that are close to the NL surface form, e.g. syntactic derivations, rather than from the surface form per se [Karttunen, 2015a; Abzianidze, 2017b]. One example of such regularities, which is also considered natural logic's most common speciality, is monotonicity. Given the argument in (1.2), for example, replacing '*car*' in p with the more specific term '*red car*' in h still preserves the truth and that is due to its occurrence in the downward-monotone context

caused by '*no*'.

(1.2)
$$\frac{p\text{: John bought } no^{\downarrow} \text{ car.}}{c\text{: John bought no red car.}}$$

Recent work in the fields of natural logic has shown that it can achieve the needed semantic precision to handle many important inference problems without the need to translate them into a formal language such as FOL. Nonetheless, natural logic-based inference systems have witnessed some shortcomings (discussed in Section 4.3), one of which, not being able to handle arguments of multi-premises, is particularly significant.

1.1 Research Motivation

Our general aim for the NLI system presented in this thesis is to reason about deep semantic phenomena (e.g. *defaults*, *monotonicity*, *quantifiers* and *propositional attitudes*), and thus construct a MR that is able preserve the fine semantic distinctions humans make when reasoning about such phenomena [Ramsay, 2009]. For decades, the best way for obtaining such semantically deep MRs was by translating NL texts into logical formulas. However, as noted above, such translations have proved difficult and have been deemed unsuccessful in wide-coverage formulations of the NLI task e.g. Bos and Markert [2005, 2006].

On the other hand, natural logic approaches have shown promising results in achieving the desired semantic precision while side-stepping the difficulty of translating into logic, yet lack any proof search mechanisms that would allow chaining over multi-premises.

Not until recently (around the time this research was conducted) have researchers started to look into incorporating some proof search techniques to allow chaining over multi-premise arguments over representations that are close to NL text surface forms. To our knowledge, there is only one particular computational NLI model, the *natural tableau* of Abzianidze [2014, 2016, 2017a], that has shown that theorem provers for FOL adapted to work on MRs that are as close as possible to NL texts, are capable of handling deep inferences once they have been provided with an extended version of inference rules that are devised specifically for the application on a particular linguistic construction [Abzianidze, 2014, 2016, 2017a].

Following that line of research, we have introduced a computational model for NLI with the following aims and questions in mind:

1.2 Research Aims

A1. Investigate the use of syntactical analysis of NL texts (dependency trees in particular) as a basis for reasoning about NL.

A2. Re-visit a class of inference problems (exemplified by parts of the Fracas data-set and some others, and centred around certain deep semantic phenomena including defaults, quantifiers, and propositional attitudes) that cannot be handled by simple subsumption relations over trees, due to the lack of mechanisms for chaining over multi-premises, nor by translating into logical expressions (which have been proven difficult) **alone**, and attempt to solve them by designing and implementing a NLI system that uses a combination of a tree-matching algorithm and theorem proving.

1.3 Research Questions

Q1. Can we use representations that are close to trees obtained by standard syntactic analysis as the MRs for complex issues in semantics?

Q2. Can the target inference problems (in A2) be tackled by adapting existing inference techniques, namely natural deduction rules and natural logic containment relations, to operate directly on those representations?

1.4 Our NLI system Architecture and Corresponding Research Contributions

To answer the above research questions, we have designed and implemented a NLI system (its general architecture is outlined in Figure 1.2) that has the following main parts (marked on Figure 1.2):

S1. A **parser** that takes a NL utterance and constructs, using **a** dependency grammar, its dependency tree (Chapter 6).

S2. A **tree normalizer** that takes a dependency tree and turns it, by applying a series of transformational stages, into a form (*inference friendly forms* (IFFs)) that the theorem prover can work with (Chapter 7).

S3. A **theorem prover** that, given a query (in its IFF), attempts to prove it from any previously mentioned premise(s) (within the current discourse) and any available information in the repository, and conclude its attempt by an answer: Yes (was able to prove it), No (was able to prove its contradiction), or Unknown (was not able to prove either the query or its contradiction) (Chapter 8).

S4. A **repository of information** to be used as background knowledge (Chapter 8).

The contributions of this work are as follows:

CO1. Investigating the literature for natural logic based computational inference systems and identifying some of the research gaps (Chapter 4).

CO2. Investigating the literature for some fine-grained representational aspects that are essential for expressing the meaning of certain semantic phenomena– including defaults, quantifiers, and propositional attitudes (Chapter 5) and using them for:

- CO.2.1 deciding on the semantic features that need to be encoded on dependency trees and how we are going to encode them (Chapter 6).
- CO.2.2 defining a set of transformational stages that can be used to normalize dependency trees into IFFs (Chapter 7).
- CO.2.3 designing a number of higher-order inference rules that we hand-code into the set of available information in CO4 (Chapter 8).

CO3. Defining a matching algorithm over pairs of IFFs based on the definition of natural logic's semantic containment (Chapter 4).

CO4. Collecting or hand-coding any pre-requisite information (some of which is common knowledge, while other elements are based on our investigation in CO2) that either the matcher or the theorem prover require and making them available in a suitable format (Chapter 8).

CO5. Adapting (in Chapter 8) a proof search mechanism (an existing FOL theorem prover– namely *constructive SATCHMO*–see Section3.3) to allow chaining over multi-premises (of normalized trees) and extending it with:

- CO.5.1 some inference rules (CO4) for handling higher-order constructions (such as propositional attitudes).

CO.5.2 a matching algorithm (CO3) as the subsumption algorithm instead of straight unification.

CO6. Providing an empirical analysis (in Chapter 9) of the system's representational and inferential adequacy by evaluating it against a standard test-set for investigating deep semantic phenomena of the kind treated by this research – namely the FraCas test suite (Section 2.2.1) and compare its outcomes to the state-of-the art NLI systems.

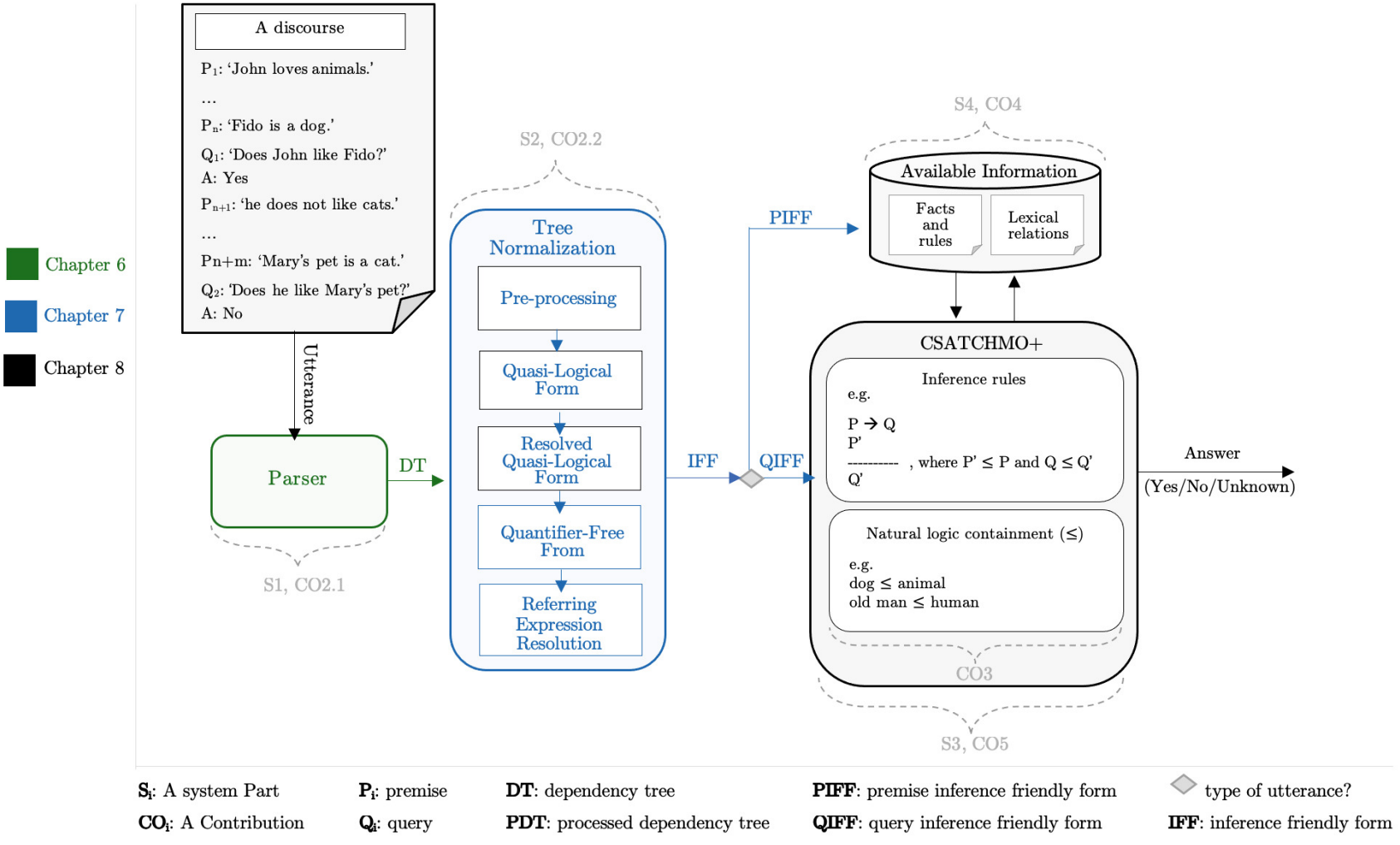


Figure 1.2: Dataflow through the inferential system.

1.5 Thesis Structure

The remainder of this thesis is grouped as following:

Chapter 1 Introduces NLI systems and the motivation, goals and questions behind the research conducted in this thesis.

Chapter 2 Briefly discusses the most prominent NLI datasets; their characteristics, merits and drawbacks.

Chapter 3 Briefly introduces the different mechanisms for FOL automated theorem proving and, in particular, explains a constructive proof system (natural deduction) as it constitutes the basis of the adapted inference engine.

Chapter 4 Introduces natural logic and monotonicity, and reviews several computational NLI systems that are based on it.

Chapter 5 Reviews a number of considerably deep semantic phenomena, particularly those of interest.

Chapter 6 Explains the first part of our inference system, which is the parser. It explains the grammar used to generate the trees and the pre-processing steps used to refine them.

Chapter 7 Illustrates the transformational steps that processed trees undergo to get turned into the forms that the inference engine desires: inference friendly forms.

Chapter 8 Explains the final part of our inferential system, which is the adapted theorem prover. It explains its proof algorithm, the data used, and the extensions to handle the intended semantic phenomena.

Chapter 9 Conducts an evaluation of our inference system's representational and inferential adequacy, and compares its performance with the related systems.

Chapter 10 Concludes the thesis with our final findings and our future plans for the presented research.

Chapter 2

The NLI Task and Related Datasets

One obvious way to test a computational model or a theory for its ability to handle NLIs is by running it against a set of NLI problems. In this chapter, we survey the most prominent NLI datasets, including the one we used for our computational system's empirical analysis (CO5) (to which we will return in Chapter 9).

2.1 Introduction

NLI, as defined in Chapter 1, is the task of determining whether a NL consequence can be inferred from a NL premise if a human would reasonably be expected to infer it. There exist several sources for inference problems that could be used for evaluating a computational model or a theory for its capability to carry out NLIs. These sources, although the above definition presents NLI as a single well defined task, formulate the NLI task quite differently [MacCartney, 2009]. One point of difference such sources might have is the number of labels used to classify the inferability of a consequent (c) from premises (p_s). Most commonly, sources are based on either the two-way or the three-way classification of *judgement*. In the first, the relationship concluded between the p_s and c is classified as either *entail* ($p_s \models c$) or *contradict* ($p_s \models \neg c$), while in the latter there is, in addition to the above two classes, a *neutral* judgement and that is when the p_s neither entail nor contradict the c (i.e. $(p_s \not\models h) \wedge (p_s \not\models \neg h)$). Other differences NLIs sources might have are the nature of their NLI problems (i.e. whether they are manually constructed, typically to explore some challenging set of phenomena or are extracted from some freely occurring texts), the number of premises each problem has and the level of their complexity, etc. While these differences play a major role in deciding what is deemed a suitable choice for evaluating a system or theory, they all to

some extent share one implicit task which is: "to answer as many problems correctly as possible" [MacCartney, 2009].

In the following section we survey the most prominent sources of NLI problems focusing on their main characteristics, merits and drawbacks. Among the discussed sources we are particularly interested in the *FraCaS test suite* (Section 2.2.1) as it is the one we used to evaluate the work presented in this thesis, however we will leave the discussion of why we chose it to Chapter 9.

2.2 NLI Datasets

2.2.1 The FraCaS test suite

The *FraCaS test suite* is a semantic test suite that was developed by the FraCaS consortium [Cooper et al., 1994] as an initial benchmark "for evaluating the inferential competence of different NLP systems and semantic theories". The test suite consists of 346 NLI problems that were constructed to cover a diverse range of deep semantic phenomena (Table 2.1 lists these phenomena and the relevant number of inference problems), and to resemble those appearing on preliminary semantics textbooks [MacCartney and Manning, 2007]. The problems are therefore mainly concerned with quantificational and logical phenomena, consist of fairly short sentences and require limited background knowledge to solve them [Chatzikyriakidis et al., 2017].

Each FraCaS problem has one or more premises, one Yes/No question and a gold answer. Most commonly, an answer is either: YES, NO, or DON'T KNOW and that is when the relation between the given premise(s) and the proposition carried by a question is either entail, contradict, or neutral respectively (see Table 2.2 for some samples) [Abzianidze, 2016, 2017b]. The ratio of these answers are not balanced; more than half of the problems have YES as their answer, 27% have DON'T KNOW as their answer and 9% have been answered with NO [MacCartney, 2009]. On the less common side, the answers of some FraCaS problems have been provided with comments that either further explain them or place some restrictions on them such as the answers of FraCas-16 and FraCas-61 in Table 2.2 respectively. Problems with such answers are often omitted on the evaluation of NLI systems that are based on the 3-way classification of judgement as non of these answers map into entail, contradict, nor neutral.

Section #	Semantic Phenomenon	Number of Problems
1	Quantifiers	80
2	Plurals	33
3	Anaphora	28
4	Ellipsis	55
5	Adjectives	23
6	Comparatives	31
7	Temporal	75
8	Verbs	8
9	Attitudes	13
	Total	346

Table 2.1: Semantic phenomena and related number of problems [Cooper et al., 1994]

The upside of the FraCaS test suite is that due to the diversity of semantic phenomena it covers and the nature of its problems, the suite is often considered to be the best when it comes to testing the expressiveness and deep reasoning capability of a semantic theory and its associated system [Abzianidze, 2017b; Bernardy and Chatzikyriakidis, 2017]. The suite also has the advantage of multilinguality [Chatzikyriakidis et al., 2017], as it was later extended into other languages including: 1) German, Farsi, Mandarin, and Greek by the *MultiFraCaS project*[1]; 2) Japanese (*JSeM*)[2]; and 3) French [Amblard et al., 2020].

On the downside, the test suite is fairly small. In addition, the problems are rather artificial (they do all represent genuine issues in semantics, but the wording of some of the examples is slightly odd) and some of the Gold answers are debatable[3]. Thus, it is believed that while the FraCaS test suite covers a wide range of challenging phenomena, it is not suitable for testing systems that aim to produce wide coverage with shallow inference, in particular examples that involve lexical relations such as hyponymy and antonymy [Abzianidze, 2017b].

2.2.2 The RTE Challenges

Another "formulation of the NLI task is the *Recognising Textual Entailment* (RTE) challenge" [MacCartney, 2009]. Starting in 2005, the challenge was held yearly and

[1] https://gu-clasp.github.io/multifracas/

[2] https://github.com/DaisukeBekki/JSeM

[3] There are a number of identical FraCaS problems that have different answers such as FraCaS-87 and FraCaS-88 in Table 2.2.

FraCaS-16 P1: At most two tenors will contribute their fees to charity.	
Q: Are there tenors who will contribute their fees to charity?	[At most two]
FraCaS-28 P1: Few committee members are from Portugal P2: All committee members are people. P3: All people who are from Portugal are from southern Europe.	
Q: Are there few committee members from southern Europe?	[DON'T KNOW]
FraCaS-61 P1: Both female commissioners used to be in business.	
Q: Did both commissioners used to be in business?	[YES, if both commissioners are female; otherwise there are more than two commissioners.]
FraCaS-81 P1:Smith, Jones and Anderson signed the contract.	
Q:Did Jones sign the contract?	[YES]
FraCaS-87 P1:Every representative and client was at the meeting.	
Q:Was every representative at the meeting?	[YES, on one reading]
FraCaS-88 P1:Every representative and client was at the meeting.	
Q:Was every representative at the meeting?	[DON'T KNOW, on one reading]
FraCaS-119 P1: No student used her workstation. P2: Mary is a student.	
Q: Did Mary use a workstation?	[NO]

Table 2.2: Samples of the FraCaS problems [Cooper et al., 1994].

has eight releases; the first three were published by PASCAL[4], whereas NIST[5] published the remaining five. In each release, contestants were provided with a set of NLI problems for testing, and (in most releases) they were also given a large development set (see Table 2.3 for some statistics) [MacCartney, 2009; Bentivogli et al., 2017].

Each RTE problem consists of two text snippets, a premise (P) and a hypothesis (H), and a judgement label. Similar to FraCaS problems, the labels were mainly either {YES} indicating that ($P \models H$), or {NO} and that is when ($P \models \neg H$). On later releases

[4] http://www.pascal-network.org/?q=node/15

[5] National Institute of Standards and Technology, http://www.nist.gov/tac/2009/RTE/

of the test set, starting from the fourth one, a third answer {UNKNOWN} was added, indicating a neutral judgement for the entailment relation between a *P-H* pair.

RTE challenges share the same broad goal as the FraCaS, namely testing the inferential capacity of NLP models [Poliak, 2020]. However, unlike the FraCase test suite, RTE problems focus on evaluating models for distinct forms of inference (e.g. *relation extraction* and *paraphrasing*) instead of specific semantic phenomena [MacCartney and Manning, 2007; Poliak, 2020]. Hence they were chosen from real sources, such as newswire texts and existing NLP datasets, instead of being constructed.

Extracting RTE problems from real sources "remedies the unnaturalness of constructed examples" [Chatzikyriakidis et al., 2017]. The RTE problems tend to be fairly long (with possibly more than one sentence) and more syntactically complex than the FraCaS examples [MacCartney, 2009]; see Table 2.4 for examples. Moreover, the definition of inference within the RTE platforms "allows presupposition of common world knowledge", which some find problematic [Zaenen et al., 2005; Chatzikyriakidis et al., 2017; Poliak, 2020]. For instance, judging the entailment relation of the RTE pair in number (2) from Table 2.4 depends on the prior knowledge that a president of a country is a citizen of that country as well [Poliak, 2020]. Last but not least, although the average size of the early RTE test sets is greater than the size of the FraCaS test suite, it is still considered small, especially for approaches that rely upon training a vast amount of data such as *deep learning* methods [Poliak, 2020].

Release Name	**Year**	**Authors**	**Test set size**	**Development set size**
RTE-1	2005	Dagan et al.	800 problems	576 problems
RTE-2	2006	Bar-Haim et al.	800 problems	800 problems
RTE-3	2007	Giampiccolo et al.	800 problems	800 problems
RTE-4	2008	Giampiccolo et al.	800 problems	–
RTE-5	2009	Bentivogli et al.	600 problems	600 problems
RTE-6	2010	Bentivogli et al.	19,972 problems	15,955 problems
RTE-7	2011	Bentivogli et al.	22,426 problems	21,420 problems
RTE-8	2013	Dzikovska et al.	7,093 problems	8,910 problems

Table 2.3: Eight releases of the RTE challenges' benchmarks [MacCartney, 2009; Bentivogli et al., 2017]

(1)	P: Most Americans are familiar with the Food Guide Pyramid but a lot of people dont understand how to use it and the government claims that the proof is that two out of three Americans are fat.	H: Two out of three Americans are fat.	[YES]
(2)	P: Meanwhile, in an exclusive interview with a TIME journalist, the first oneon-one session given to a Western print publication since his election as president of Iran earlier this year, Ahmadinejad attacked the threat to bring the issue of Irans nuclear activity to the UN Security Council by the US, France, Britain and Germany.	H: Ahmadinejad is a citizen of Iran.	[YES]
(3)	P: At the same time the Italian digital rights group, Electronic Frontiers Italy, has asked the nation's government to investigate Sony over its use of anti-piracy software.	H: Italy's government investigates Sony.	[NO]
(4)	P: Four people were killed and at least 20 injured when a tornado tore through an Iowa boy scout camp on Wednesday, where dozens of scouts were gathered for a summer retreat, state officials said.	H: Four boy scouts were killed by a tornado.	[UNKNOWN]

Table 2.4: Four RTE examples from RTE-1, RTE-2, RTE-3, and RTE-4 test suit respectively.

2.2.3 SICK

Marelli et al. [2014] developed the *Sentences Involving Computational Knowledge* dataset to test *Computational Distributional Semantic Models* (CDSMs). The dataset consists of 10K pairs of English sentences rich in lexical, syntactic, and semantic phenomena that CDSMs are expected to account for (e.g. quantifiers, negation, passive/active alternation, synonyms, and relative clauses) [Marelli et al., 2014; Abzianidze, 2017b; Chatzikyriakidis et al., 2017].

The 10k SICK pairs are split into 500 for trial, 4500 for training, and 4927 for

testing. SICK pairs, similar to FraCaS problems, are short in length. However, unlike FraCaS problems they were generated from real sources, captions of pictures[6] and videos[7] [Marelli et al., 2014; Abzianidze, 2017b]. Moreover, using crowd-sourcing SICK pairs were labelled with two gold answers:

1. A score (from 1 to 5) that indicates a *relatedness* in meaning between the sentences of a pair.

2. A 3-way judgement label (entail, contradict, neutral) indicating the entailment relation from the first sentence in the pair to the second.

13 examples of SICK pairs and their gold answers are listed in Table 2.5. As it can be noticed these examples are all lexically and semantically similar in some way or another. This goes back to the way they were generated. Put differently, each pair of sentences describing the same photo or video go through a series of normalization then expansion rules that in turn lead to more pairs. Expansion rules create more variants – they include, for example, a rule to replace a word with its synonym (e.g. '*A young boy is...*'→ '*A young kid is...*'), a rule to insert a negation (e.g. '*A boy is playing...*'→ '*a boy is not playing...*') or replace a determiner with its opposite creating a contradiction (e.g. '*A dog is walking...*'→ '*No dog is walking...*'), etc [Marelli et al., 2014]. Normalization rules, on the other hand, are to ensure that no generated pair involves a phenomenon that CDSMs cannot account for. This includes [Marelli et al., 2014]: 1) named entities that are normalized into words that represent their class (e.g. '*... is playing Mozart*' → '*... is playing classical music*'); 2) verb phrases with auxiliaries or modals which are turned into simpler ones (e.g '*a kid has to eat...*' → '*a kid is eating ...*'); and 3) multiword expression which are removed completely (e.g '*... is playing guitar right now*' → '*... is playing guitar*'.

The above characteristics of SICK problems make them at first glance look like logical problems and has led some researches (e.g. [Abzianidze, 2014, 2015, 2017b] and [Martínez-Gómez et al., 2017]) to use the dataset to evaluate their logic-based NLI systems. However, the SICK dataset being specifically tailored for CDSMs conflates (through normalization) some phenomena and fine details that are crucial for evaluating the semantic competence of a NLI system (e.g. losing tense information by simplifying verb constructions and getting rid of auxiliaries).

[6] http://nlp.cs.illinois.edu/HockenmaierGroup/data.html
[7] http://www.cs.york.ac.uk/semeval-2012/task6/index.php?id=data

SICK Pair		Score	Judgement
A sea turtle is hunting for fish	A sea turtle is hunting for food	4.5	Entail
A sea turtle is not hunting for fish	A sea turtle is hunting for fish	3.4	Contradict
A fish is hunting for a turtle in the sea	A sea turtle is hunting for fish	3.9	Neutral
The turtle is following the red fish	The turtle is following the fish	4.6	Entail
The turtle is following the fish	The turtle isnt following the fish	4	Contradict
The turtle is following the fish	The fish is following the turtle	3.8	Contradict
A sea turtle is hunting for fish	The turtle is following the red fish	4	Neutral
A sea turtle is hunting for fish	The turtle isnt following the fish	3.2	Neutral
The fish is following the turtle	A sea turtle is hunting for fish	3.2	Neutral
The turtle is following the fish	A sea turtle is hunting for food	3.9	Neutral
The turtle is following the fish	A sea turtle is not hunting for fish	3.4	Neutral
A fish is hunting for a turtle in the sea	The turtle is following the fish	3.5	Neutral
A sea turtle is hunting for fish	The turtle is following the fish	3.8	Neutral

Table 2.5: A 13 pairs of SICK problems with their two gold answers [Abzianidze, 2017b]

2.2.4 SNLI and MNLI

A more recent dataset for the NLI task is the *Stanford Natural Language Inference* (SNLI) [Bowman et al., 2015] and its descendant the *Multi-Genre Natural Language Inference* (MNLI) [Williams et al., 2017]. The development of SNLI came as a remedy to the lack of large-scale datasets that *machine learning* methods can use. Thus, with 570K pairs of English sentences, SNLI is considered as the largest NLI dataset of its kind.

Each SNLI pair consist of a premises P and a hypothesis H, and labelled with an entailment judgement (entail, contradict, or natural). Premises are image captions selected from the *Flickr30K* corpus [Young et al., 2014], while hypotheses are elicited (using crowd-sourcing) sentences from these caption. Put differently, from each image caption annotators were asked to write entailing, contradicting and neutral sentences [Poliak, 2020].

Despite being the first "empirical evaluation for learning-centered approaches" [Bowman et al., 2015], SNLI dataset, coming from one written genre (image captions),

falls short when it comes to exemplifying a number of important phenomena, such as propositional attitudes (e.g. know and believe), modality (e.g. must and should), and temporal adverbs (e.g yesterday) [Williams et al., 2017]. MNLI has targeted these shortcomings by developing 433k NLI problems in a similar fashion as the SNLI data set, but from ten distinct genres (e.g. Government, Travel and 9/11) of spoken and written English, see Table for examples from both datasets.

SNLI Pair		**Judgement**
P: A soccer game with multiple males playing.	H: Some men are playing a sport.	Entail
P: A black race car starts up in front of a crowd of people.	H: A man is driving down a lonely road.	Contradict
P: An older and younger man smiling.	H: Two men are smiling and laughing at the cats playing on the floor.	Neutral
MNLI Pair		**Judgement**
P: At 8:34, the Boston Center controller received a third transmission from American 11	H: The Boston Centre controller got a third transmission from American 11.	Entail
P: Met my first girlfriend that way.	H: I didnt meet my first girlfriend until later.	Contradict
P: I am a lacto-vegetarian	H: I enjoy eating cheese too much to abstain from dairy.	Neutral

Table 2.6: Some NLI problems from the development set of the SNLI (top) [Bowman et al., 2015] and MNLI (bottom) [Williams et al., 2017] along with their gold judgement.

2.3 Summary

In this chapter we have surveyed the five most commonly used NLI dataset which are: the FraCaS test suite, the RTE datasets, the SICK dataset, and SNLI and MNLI datasets. The goal of that brief survey is to present the necessary information that will help the reader to follow:

- Our discussion (in Chapter 4) of state-of-the art NLI systems we compared our work to.
- Our justification (given in Chapter 9) of why FraCaS problems might be the best choice for evaluating our computational system for its capability to handle the NLIs that it is designed for.

Chapter 3

A Constructive Theorem Prover

A major task we want to implement in our NLI system is the ability to reason over multi-premises. Having a theorem prover (a proof search mechanism) as a part of the system will allow us to do that. Therefore, after a brief introduction about theorem proving (Section 3.1), in this chapter we illustrate the theorem prover we are adapting (CO5) in this thesis (Section 3.3) and explain the proof system that inspired its rules of inference (Section 3.2).

3.1 Automated Theorem Provers

In mathematics, a *theorem* can be defined as a non-self evident statement whose truth was proved based on already established truths (other theorems or axioms) in accord with some deductive rules. A computer program used to prove a theorem is called an *automated theorem prover*. For formal languages such as propositional logic and FOL, there exist many theorem provers. These provers can be divided into different families of proof mechanisms including: (1) *refutation* systems such as tableau and resolution that prove a formula by proving that its negation leads to a contradiction, (2) *axiomatic* systems which rely on a predefined set of axioms and derivation rules (e.g. Hilbert system); and (3) *constructive* systems such as natural deduction which builds its proofs constructively from assumptions and already completed proofs. It has been claimed that natural deduction "formalizes the kind of reasoning people do in informal arguments." [Fitting, 1990, p. 77]. If that is true, it seems plausible that natural deduction constructiveness may fit well with reasoning about natural language.

3.2 Natural Deduction

Natural deduction (ND) is a proof system that tries to prove valid propositions by repeatedly applying simple basic inference rules and possibly introducing some valid assumptions. These rules can be seen as an interpretation for the logical constants–such as conjunction, disjunction, implication and universal quantifier–which, in a sense, is said to make them *natural*, as they closely correspond to the common steps in intuitive reasoning [Prawitz, 1965]. Also, it is important to know that some of the ND rules can be combined together practically during the course of proof. An important case of such rule combination is what leads to the notion of *unification* in proofs, and that in turn gives a kind of flexibility in allowing the use of two or more non-identical propositions, textitP and P', in a deduction, so long as they are unified. ND allows the introduction of assumptions during the course of proof and some constraints for discharging these assumptions afterwards.

3.2.1 History

Natural deduction (ND) was formally proposed in 1934, independently, in publications by the logicians Gentzen [1934] and Jaśkowski [1934], out of dissatisfaction with Hilbert-style axiomatic systems in terms of constructing mathematical proofs. Many researchers, including Gentzen and Jaśkowski, believed that, despite the preciseness of the theoretical proofs offered by the common logical systems at that time – systems of Hilbert, Frege, Russell, and others – in practice, they were quite complex and artificial, as the whole concept of intuitive reasoning had drifted away [von Plato, 2014; Indrzejczak, 2010]. In 1926, Łukasiewicz brought this issue up in a Warsaw seminar, in which he explained that what mathematicians informally do when constructing their proofs does not involve using predefined axioms. Instead, they tend to make use of other methods, in particular, introducing assumptions and observing what they could lead to [Pelletier and Hazen, 2012]. Łukasiewicz was questioning the possibility of a logical theory that could adopt this idea but still be able to prove theorems that existing logical systems at that time could prove. The term '*natural deduction*' was first used by Gentzen for his logical theory. However, Pelletier and Hazen [2012] argue that Jaśkowski was the first to respond to Łukasiewicz's concerns by publishing the first non-axiomatized system that embodied these ideas in 1929 at the First Polish Mathematical Congress [Prawitz, 1965], and then publishing the refined version of his proof

system under the name '*method of suppositions*,' in 1934. The two founders' descriptions of what is known as ND differ in many aspects, such as the kind of rules and the type of proof representation. Nonetheless, they carry similarities that are considered nowadays the distinguishing features of ND systems (Section 3.2.2).

3.2.2 Natural Deduction Features

There is no single defining description of ND systems; instead, there are certain features that usually form part of what can be called a ND system [Pelletier and Hazen, 2012]. According to the remarks of the main inventors, Gentzen and Jaśkowski, as well many elementary logic textbooks, a ND system ought to have at least the following features.

Hypothetical and parametric judgements: Among the different forms of judgements, *hypothetical* judgements (conditionals) and *parametric* judgements (in the case of quantification), are the norm for ND. Hypothetical judgements are usually used for proving statements of the form, $P \rightarrow Q$ by showing that Q can be derived from an assumption P. Derivations from assumptions can be either direct or indirect. In other words, an assumption could lead to the conclusion directly or trigger another assumption, constructing a series of sub-derivations that all constructively lead to the conclusion. For that reason, a means of tracking or a bookkeeping device is required to ensure that such assumptions are introduced and discharged properly during the derivation process. For example, one of Jaśkowski's methods of doing such tracking is graphical: drawing boxes around each assumption course and, in the case of nested sub-proofs, completing the inner boxes before the outer ones, such that the inner sub-proofs can inherit any given statements from the outer sub-proofs, as for example in Figure 3.2. The format used in the following illustrations is similar to Jaśkowski's, with square brackets, i.e. [assumption], used instead of rectangular boxes to distinguish an assumption from other premises. Above this will be a labelled horizontal line to indicate the beginning of the assumption's scope and another one with the same label after the last line of the assumption's scope, i.e. $\overline{[assumption]}^{\text{label}}$ (Figure 3.1). It is important to know that an assumption may not need to be used in a derivation, may be used more than once or may be used exactly once as in *linear logic* [Girard, 1987, 1995, 1998].

Parametric judgements are basically related to the logical quantifiers Q, universal and existential, for which a judgement $Qx.P(x)$ is valid if the substitution of x by arbitrary ground term t (Q is universal) or a new constant c (Q is existential) leads to a

valid conclusion.

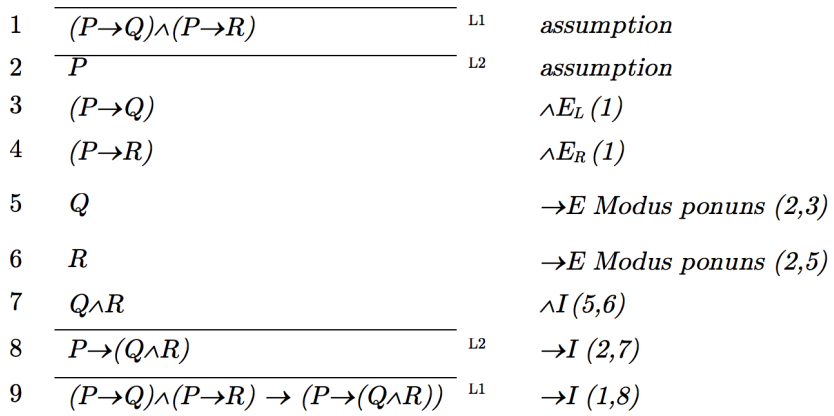


Figure 3.1: Example of proof construction.

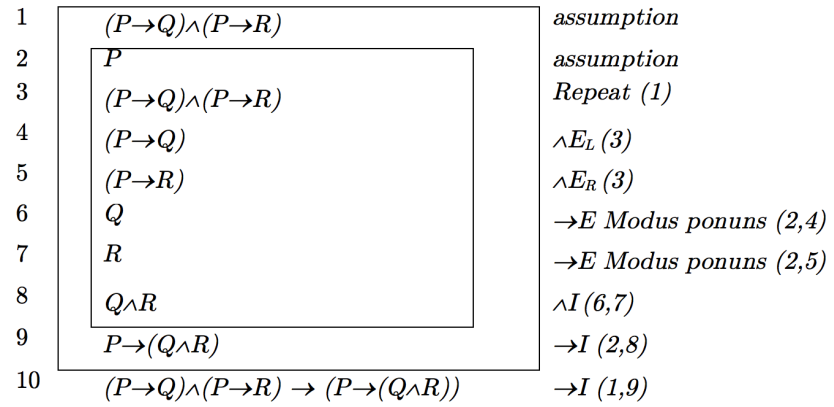


Figure 3.2: Example of Jaśkowski's graphical representation.

Inference rules: A ND system consists of simple and self-evident inference rules. These rules actually act as a defining tool for logical connectives that is given in terms of their proof rules only without reference to other logical connectives. Each of the logical connectives C, has introduction (C_I) and elimination (C_E) rules. Based on Prawitz [1965], the introduction rule for a connective C allows deduction "*to* a formula", of which C is a part, whereas, the elimination rule allows deduction "*from* a formula", of which C is a part.

- Conjunction ($\wedge$): from any two true formulas P and Q, $(P \wedge Q)$ is deducible by means of the $(\wedge_{\mathrm{I}})$ rule. On the other hand, from $(P \wedge Q)$ P and Q can be deduced using the rules $(\wedge R_{\mathrm{E}})$ and $(\wedge L_{\mathrm{E}})$ respectively.

$$\frac{P \qquad Q}{P \wedge Q}\,(\wedge_{\mathrm{I}}) \qquad\qquad \frac{P \wedge Q}{P}\,(\wedge R_{\mathrm{E}}) \qquad\qquad \frac{P \wedge Q}{Q}\,(\wedge L_{\mathrm{E}})$$

- *Disjunction* $(\vee)$: for any two formulas P and Q, $(P \vee Q)$ is deducible by means of the $(\vee_{\mathrm{I}})$ rule, if at least one of them is true.

$$\frac{P}{P \vee Q}\,(\vee L_{\mathrm{I}}) \qquad\qquad \frac{Q}{P \vee Q}\,(\vee R_{\mathrm{I}})$$

  On the other hand, using the $(\vee_{\mathrm{E}})$ rule on the true formula $(P \vee Q)$ does not allow the derivation of P or Q directly, because it is not clear whether the truth of P, Q, or both was the reason for the disjunction truth. Hence, disjunction elimination usually proceeds by finding some R that is implied by both P and Q; and this is called *proof by case*. In other words, R will be derived hypothetically as a conclusion by showing that it is derivable under the assumption of P and under the assumption Q too.

$$\frac{\begin{array}{ccc} & \overline{[P]}^{\mathrm{L1}} & \overline{[Q]}^{\mathrm{L2}} \\ & \vdots & \vdots \\ P \vee Q & R & R \end{array}}{R}\,{\scriptstyle \mathrm{L1}\,,\,\mathrm{L2}}\,(\vee_{\mathrm{E}})$$

- *Implication* $(\rightarrow)$: $(\rightarrow_{\mathrm{I}})$ rule is basically what was explained previously as hypothetical derivation.

$$\frac{\begin{array}{c} \overline{[P]}^{\mathrm{L}} \\ \vdots \\ Q \end{array}}{P \rightarrow Q}^{\mathrm{L}} \quad (\rightarrow_{\mathrm{I}})$$

  The $(\rightarrow_{\mathrm{E}})$ rule is actually nothing but the *modus ponens* rule; given that $P \rightarrow Q$ and P are true, Q can be deduced as a conclusion.

$$\frac{P \to Q \qquad P}{Q} \quad (\to_{\mathrm{E}})$$

- *Negation*$(\neg)$: In intuitionist logic, $\neg P$ is treated as implication to absurd or falsehood $P \to \bot$. Therefore, the $(\neg_{\mathrm{I}})$ rule is the same as the $(\to_{\mathrm{I}})$ rule, in which P will be assumed and if it leads to $\bot$, then $\neg P$ is a valid conclusion.

$$\frac{\begin{array}{c}\overline{[P]}^{\mathrm{L}} \\ \vdots \\ \bot\end{array}}{\neg P}\,\mathrm{L} \quad (\neg_{\mathrm{I}})$$

  In Gentzen's ND for classical logic different rules have been introduced as a $(\neg_{\mathrm{E}})$. One of them is the *Reduction ad Absurdum* rule:

$$\frac{P \qquad \neg P}{\bot}\ (\bot_{\mathrm{E}})$$

- *Universal quantification* $(\forall)$: The universal introduction rule tries to generalize the proposition *P(a)* from being about the parameter a to every x in the domain of discourse; $(\forall x.P(x))$. This is possible if this generalisation would lead to the same truth that *P(a)* lead to when used as a line of a proof, where none of the assumptions nor conclusion has an occurrence of the parameter *(a)*. Just then we can say:

$$\frac{P(a)}{\forall x.P(x)}\,\mathrm{a}\ (\forall_{\mathrm{I}})$$

  Contrariwise, knowing that P is true for every x of the domain of quantification, means that substituting x by any t from that domain, $P(t/x)$, is also true.

- *Existential quantification* $(\exists)$: The $(\exists_{\mathrm{I}})$ rule states that if there is at least one term t that could substitute x in *P(x)* and makes P true, then the truth of $\exists x.P(x)$ as a conclusion is guaranteed. However, the $(\exists_{\mathrm{E}})$ rule is not as straightforward as

$$\frac{\forall x.P(x)}{P(t)}\ (\forall_{\mathrm{E}})$$

the $(\exists_{\mathrm{I}})$ rule, because knowing $\exists x.P(x)$ is true does not mean that $P(t/x)$ could be true for any t. Because it is a rule of existential quantification, it holds only for a certain *t(s)* that we do not know. Therefore, to extract a consequence from $\exists x.P(X)$, a substitution $P(a/x)$ for a 'new' parameter a will be introduced as an assumption and see if it will lead to a valid conclusion.

$$\frac{P(t)}{\exists x.P(x)}\ (\exists_{\mathrm{I}})$$

$$\frac{\begin{array}{c}\overline{P(a/x)}^{\mathrm{L}} \\ \vdots \\ \exists x.P \quad R\end{array}}{R}\,{}_{\mathrm{L,a}} \qquad (\exists_{\mathrm{E}})$$

As may be seen, in a sense, each elimination rule is the inverse of its corresponding introduction rule. In application, this could lead to a nonsense detour; in a proof, there is nothing to gain if a premise is used to construct a formula by means of a proper introduction rule, then used as a major premise for the corresponding elimination rule, only to obtain the starting premise again. Therefore, theorem provers tend to perform a kind of reduction into normal forms to avoid such detours. This is referred to as the *inversion principle* [Prawitz, 1965]. Also, it is important to know that in practice, some of the ND rules can be combined together to form a new rule(s). An important case of such a rule is the combination of the $\forall_{\mathrm{E}}$ and $\rightarrow_{\mathrm{E}}$ rules, which leads to the notion of *unification* in proofs (Figure 3.3). By means of this combined rule, we can say that Q' is a valid conclusion from $P \rightarrow Q$ and P' if P and P', Q and Q' are unified $(\oplus)$ using the same unifier.

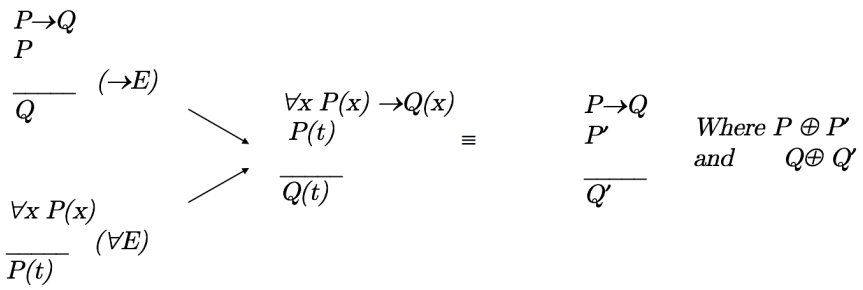


Figure 3.3: A combined rule.

3.3 SATCHMO as a Natural Deduction Engine

SATCHMO (*'SATisfiability CHecking by MOdel generation*) is an automated theorem prover for the language of FOL written in Prolog that was introduced by Manthey and Bry in [1987][1988]. Although their model generation paradigm was first motivated by logic databases problems, it handled, with great efficiency, many of the theorem proving problems discussed in the literature.

SATCHMO accepts set of FOL clauses of the form $(\neg A_1 \vee \ldots \neg A_n \vee C_1 \vee \ldots C_m)$[1] as specification but in implication form $(A_1 \wedge \ldots A_n \rightarrow C_1 \vee \ldots C_m)$. The left-hand side (LHS) of the rule is called the antecedent and it can have up to *n* literals or the constant *'true'*, which in this case the rule is actually called a *fact* or *statement*. On the other hand, the consequent – the right-hand side (RHS) – of the rule can have up to *m* literals or the constant *'false'* and in the case of the later it is called an *integrity constraint* (see below).

Definition 2.1: A model *M* of a set of clauses *S* is the set of the ground literals that are satisfied in the model. A clause *X* is said to be satisfied in the model [Manthey and Bry, 1988]:

1. if it is a ground literal and *M* contains it.
2. if it is a ground conjunction/disjunction and *M* contains all/some of its components.

[1] **In FOL**:

- a *clause* is disjunction of literals.
- a *literal* is an atomic formula or its negation.
- a *ground literal* is a literal with no variables.
- an *atomic formula* is one with no logical connectives nor quantifiers.

3. if it is a clause ($A \rightarrow C$) and $C\sigma$ is satisfied in M (for every substitution σ) such that $A\sigma$ is satisfied.
4. if it is a clause ($A \rightarrow C$) and there is a substitution σ such that $A\sigma$ is satisfied and $C\sigma$ is not, the clause X can be satisfied by asserting:
   (a) $C\sigma$ if it is an atom;
   (b) a component from $C\sigma$, if it is a disjunction and create a choice point for backtracking.

SATCHMO is a two-way prover, as it exploits for the above definition two kinds of proof mechanisms. Prolog's backward chaining mechanism for handling *horn-clauses* (1-3) (clauses that consist of at most one positive literal), and an introduced second type of rules to be applied forward[2] for the *non-horn* ones (4) [Loveland et al., 1995].

```
satisfiable :-
        is_violated(C), !,
        satisfy(C),
        satisfiable.
satisfiable.

is_violated(C) :-
        (A ---> C),
        A, not C.


on_backtracking(X).
on_backtracking(X) :-
        X, !, fail.
```

```
satisfy(C) :-
        component(X,C),
        asserta(X),
        on_backtracking(retract(X)),
        not false.


component(X,(Y;Z)) :-
        !, (X=Y ; component(X,Z)).
component(X,X).
```

Figure 3.4: The model generation process [Manthey and Bry, 1988].

As Figure 3.4 shows, SATCHMO is a model generation and a refutation system, which means two things. First, as a refutation system, it performs its proofs by contradictions, i.e. it proves that a clause C is satisfied in every M by showing that $\neg C$ produces a contradiction (is_violated(C)). Second, it uses violated clauses of kind (4) as a *generation rules* to extend the current model using (satisfy(C)). What (satisfy(C)) does, as stated in (4), is assert the violated consequent itself or one of its components

[2]Reasoning forward means starting with known facts and follow it by sequence for rules application to obtain more data, until the goal is reached. It often considered as a repeated application of the *modus ponens* rule; use the known data to see if there is ůle that matches its antecedent and add its consequent. Backward reasoning, on the other hand, starts with the goal and chains backwards looking for facts that support the goal; look for rules whose consequent matches the goal and then try proving their antecedent [Polleres et al., 2011, p .327].

and make a choice point in the case of the latter. If an asserted consequent generated a 'false', its assertion has violated the integrity constraints and has to be retracted and everything added since its assertion is backtracked. If a created model is not empty and in which every clause in S has been satisfied, it said that the satisfiability of that set of clauses had been shown with respect to that model. For example, the clause set in Figure 3.5 has generated an empty model (all choice branches generated false) and hence the clause set under consideration is unsatisfiable.

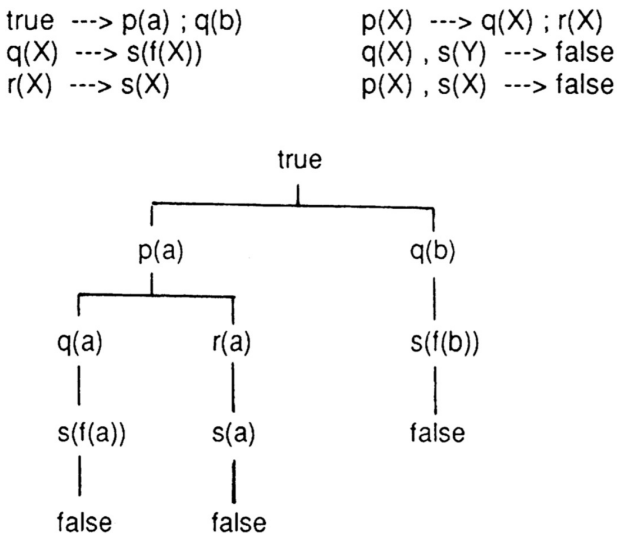


Figure 3.5: Example of model Generation in Prolog [Manthey and Bry, 1988].

A major advantage of SATCHMO is that its implementation is short and simple (Figure 3.4), which makes its adaptation to different applications and non-standard treatments of a language, such as defaults and intensionality, quite easy. One adaptation is Ramsay's [2001] *constructive SATCHMO*. As the name implies, the core change is that in contrast to SATCHMO, it does its proof constructively using ND rules. Table 3.1 shows the constructive SATCHMO equivalent rule for the implication rule. The way in which its rule embodies the constructive rule of implication is by using `assert(P)` to put something as a temporary assumption and then try to `prove(Q)` from it. Finally, making sure that the asserted assumption is removed after the end of the proof course whether or not the proof succeeded, by using `retract(P)`. Beside classical ND rules, Ramsay's constructive SATCHMO has some higher-order extensions. For the sake of the aims of this thesis, we are adopting only its basic rules (given in Figure 3.6) with our extension that is suitable for the work on normalized dependency trees. Our adapted inference engine will be further explained in Chapter 8 when

the relevant concepts have been already discussed.

$\dfrac{\begin{array}{c}\overline{[P]}^{\mathrm{L}} \\ \vdots \\ Q\end{array}}{P \to Q}^{\mathrm{L}} \quad (\to \mathrm{I})$	`Prove (P→Q):-`
	`  assert(P),`
	`  (prove(Q)→retract(P);`
	`  (retract(P), fail))`

Table 3.1: ND and Ramsay's constructive SATCHMO treatment for implication.

```
% You can prove A either directly
prove(A) :-
    A.

% or by proving (P or Q), (P => A) and (Q => A)
prove(A) :-
    split(P; Q),
    \+ (P; Q), % check you haven't tried this already
    prove(P => (A or absurd)),
    prove(Q => (A or absurd)).

% To prove (P => A), assert P and try to prove A
% (with some funny bookkeeping to tidy up after yourself)
(P => A) :-
    assert(P),
    (prove(A) -> retract(P); (retract(P), fail)).
```

Figure 3.6: Basic constructive SATCHMO [Ramsay, 2001].

3.4 Summary

One goal of this thesis is to be able to answer multi-premises NLI problems. Having a theorem prover as a proof search mechanism is an obvious way to do this.

In this chapter we have discussed what is a theorem prover and what are the different kinds of proof mechanisms (refutation, axiomatic and constructive) for formal languages such as FOL and prepositional logic. Included in this discussion, we have explained at considerable length constructive theorem proving (natural deduction system in particular) as it is believed to be a formalization to the kind of reasoning humans do. Because of that and because of its short and simple implementation, we are adapting and extending Ramsay's [2001] constructive SATCHMO to work on normalized tress instead of FOL formulae. We will return to this in greater detail in Chapter 8.

Chapter 4

Natural Logic and Monotonicity

Natural logic is a proof system which, in contrary to the proof systems discussed in (Chapter 3), explains inferences occurring in NL with no appeal to logical formalism or model theory [MacCartney and Manning, 2007]. Instead, it relies upon representations of NL texts that are close to their surface form; most commonly syntactical trees.

One of the aims of this thesis is to investigate the choice of MRs that are based on syntactical trees. We also aim to define an algorithm for matching trees (CO3) that is based on the semantic and syntactic aspects (namely the containment relation and polarity marking respectively) of one of natural logic's most significant notions: monotonicity. Therefore, for the sake of comprehension and preparing the ground for later discussions, the first part of this chapter briefly introduces natural logic (Section 4.1) and monotonicity (Section 4.2).

In this thesis, we also compare our final system (CO6) to other NLI systems that were based on natural logic. Thus, the second part of this chapter (Section 4.3) surveys the most recent inference systems that have been based on it and point out some of the research gaps that we have identified (CO1).

4.1 Introduction

Natural logic is a term that was coined by Lakoff [1972] and generally refers to the regularities that can prove a valid NL argument by operating on representations that are close to the NL surface from, e.g. syntactic derivations, unless from the surface per se [Abzianidze, 2017b]. For example, the argument in (4.1) is valid because the replacement of '*men*' in p with '*athletic men*', which is smaller in meaning (i.e. its denotation can be contained by the denotation of '*men*'), in h is licensed by the *downward-monotone* context (labelled by $\downarrow$), in which '*men*' appears. On the other

hand, the *upward-monotone* context (labelled by $^{\uparrow}$) that contains '*racing cars*', allows its replacement with the larger term '*car*'. These licensing contexts are caused by a semantic property named *monotonicity* of the quantifier '***All***' that is downward-monotone on its first argument and upward-monotone on it second. Now, if we consider '*men*' in 4.2, it can not be replaced, it needs an exact match and this is due to its occurrence in a *non-monotone* (enclosed by $\not\uparrow$) context that is caused by '***Most***' (discussed further in Section 4.2).

(4.1) $$\frac{p : \textbf{All}\ (men)^{\downarrow}\ (love\ some\ racing\ cars)^{\uparrow}.}{c : \text{All athletic men love some cars.}}$$

(4.2) $$\frac{p : \textbf{Most}\ (men)^{\not\uparrow}\ (love\ some\ racing\ cars)^{\uparrow}.}{c : \text{Most men love some cars.}}$$

$$\frac{\text{All X Y}}{\text{All X }'\text{Y}'.}(\text{All}^{\downarrow\uparrow}) \qquad \text{s.t. } (\text{X}'\leq\text{X and Y}\leq\text{Y}')$$

$$\frac{\text{Most X Y}}{\text{Most X Y}'.}(\text{Most}^{\not\uparrow\uparrow}) \qquad \text{s.t. } (\text{Y}\leq\text{Y}')$$

Table 4.1: An illustration to what might be considered a natural logic rule in which $(\leq)$ is the order-relation defined and explained in Section 4.2.

If the regularities licensing the above valid inferences are to be formed as rules, they will be similar to what is depicted in Table 4.1, which is to its first approximation much like an extended version of the rules appearing in Aristotelian classical logic [Moss and Wollowski, 2017]. Moreover, natural logic avoids the full translation into logical formulas and makes no appeal to models.

Natural logic is not a contemporary concept. It has a long history[1], originating in ancient times, in Aristotelian syllogisms to be exact, as an attempt to simplify and generalise the ad hoc inference rules (syllogisms) [Ludlow, 2002] and expand their coverage to something more than a fragment of English. The idea at that time was to try to fit these rules into two distinct inference paradigms, *dictum de omni* (upward monotone) and *dictum de nullo* (downward monotone). These two paradigms belong to a theory that was developed by mediaeval logicians called the *Doctrine of Distribution* [Benthem et al., 2008]. However, it can be said that the *dictum de omni et nullo* paradigms have been revived in recent times as *upward-monotone* and *downward-monotone* entailing contexts in *monotonicity calculus* [Sánchez Valencia,

[1] A more detailed article on the history of natural logic is given in [Benthem et al., 2008].

1991; Van Benthem, 1986]. Since then, their work has been the basis of most recent successful work using natural logic in the field of NLI .

Natural logic has been associated mainly with reasoning about monotonicity now as then, and we will continue its discussion in those terms in the following section. Nonetheless, it is believed [Bernardi, 1999; Van Eijck, 2005] that it covers different forms of NLIs of different complexities with monotonicity as just a part of that enterprise. That becomes clear in the discussion of of natural logic based inference systems in Section 4.3.

4.2 Monotonicity Calculus

4.2.1 Monotonicity Reasoning

Reasoning about monotonicity, such as in 4.1 and 4.2, is concerned with the ability to replace an expression by another one whose denotation can contain (be expanded to) or be contained by (be contracted to) the denotation of the original expression and **still preserve truth** [Icard III and Moss, 2014; Bernardi, 1999]. In model-theory, the denotations of NL expressions are sets of objects in the domain of discourse. The sets are usually assumed to be **partially ordered** [Icard III, 2012; Fyodorov et al., 2003], and hence, a relation holding between expressions denotations of the kind described above is expressible in terms of the *order-relation*($\leq$) and is often called the *semantic containment* relation. It is defined over expressions of different types (not only sentences), and what determines if a containment relation between two expressions (of the same type) is *preserved*, *reversed*, or *cancelled* when appearing in the context of another expression, is the monotonicity property of that context. Now considering a containment relation between the denotation of two expressions of the same type, $E_1 \leq E_2$, and a context M, monotonicity inference schemes can be generally defined as follows:

Definition 3.1 [Bernardi, 1999] if the context M is:

1. *upward-monotone (=monotone)*($\uparrow$) then $M(E_1) \leq M(E_2)$, and $\frac{M\,E_1}{M\,E_2}(\mathrm{M}^{\uparrow})$ is a valid inference.

2. *downward-monotone (=antitone)*($\downarrow$) then $M(E_2) \leq M(E_1)$, and $\frac{M\,E_2}{M\,E_1}(\mathrm{M}^{\downarrow})$ is a

valid inference.[2]

3. *non-monotone*($\not\uparrow\not\downarrow$) then neither 1 nor 2 applies.

Consider for example the expressions '*red car*' and '*car*'. As they stand, the meaning of the latter contains the meaning of the first and hence '*red car*'$\leq$'*car*'. However, when these two expressions appear in the antitone context caused by the negation word '*no*' in 4.3, the containment relation is reversed; '*no car*'$\leq$'*no red car*' and hence the sentence containing '*no car*' entails the one with '*no red car*' and not the other way round. Now that we know in which context expansions or contractions of expressions could lead to valid inferences, the questions remain as to what forms they take and what sets the context.

(4.3) $$\frac{p : \text{John bought } \textbf{no } (car)^{\downarrow}}{c : \text{John bought no red car}}$$

Replacing an expression by its expansion or contraction can be seen as a **sequence** of edits, including substitution, insertion and deletion with the edit sequence of an expression yielding a valid inference if each of the single edits (linking p and c) obeys the monotonicity property of the context in which it appears. For example, (4.4) lists some valid monotone inferences in which (a) involves substituting '*man*' with the larger in meaning word: '*human*' and (b) expanded '*a fat man in the park*' to '*man* ' by deleting the intersective modifier '*fat*' and prepositional phrase.

(4.4) a. $$\frac{p : (He)^{\uparrow} \textbf{ saw } (a\ man)^{\uparrow}.}{c : \text{He saw a human.}}$$

b. $$\frac{p : (He)^{\uparrow} \textbf{ saw } (a\ fat\ man\ in\ the\ park)^{\uparrow}.}{c : \text{He saw a man.}}$$

On the other hand, (4.5) shows examples of inferences in an antitone context caused by the quantifier '*no*'. In (a) '*likes*' is substituted with the more specific term '*loves*', while in (b), '*woman*' is contracted by specifying a descriptive detail '*smart*' and hence become '*a smart woman*'.

(4.5) a. $$\frac{p : \textbf{No } (man)^{\downarrow}\ (likes\ a\ woman)^{\downarrow}.}{c : \text{No man loves a woman.}}$$

b. $$\frac{p : \textbf{No } (man)^{\downarrow}\ (likes\ a\ woman)^{\downarrow}.}{c : \text{No man loves a smart woman.}}$$

[2]Note that the inverse of $\leq$ in some publications is represented as $M(E_1) \geq M(E_2)$ and named *inversion* relation. However, in this chapter we tried to keep relational symbols to the minimal to avoid confusion.

As to what determines the nature of a context, one observation about these examples is that expressions, particularly of functional-type (we will see what that means shortly), are the ones affecting the monotonicity of their arguments (if any). For instance, we can see in 4.4 that the transitive verb '*saw*' takes two arguments and has a (default) monotone effect on both of them. By contrast, the quantifier '*no*' marks both of its arguments with an antitone property. Given a grammar, a *function-expression* is an element in the lexicon of that grammar and is to be applied to argument(s) of certain type(s) (determined by the function type itself) yielding a result of another. Put differently, consider Montague-style types [Montague, 1973, 1974], expressions of functional types ($\langle\alpha,\beta\rangle$) are those taking argument of type α and yielding a result of type β, such as the intransitive verb '*walks*' of type $\langle e,t\rangle$, that takes the proper name '*John*' of type e as argument, and provides '*John walks*' as the result (which is of type t).

The monotonicity property of a function-expression to its argument can be considered as an instantiation of the property of the mathematical mapping function f, between two sets, α and β, with partial orders $\leq_\alpha$ and $\leq_\beta$, respectively, that is defined as follows:

Definition 3.2 [Bernardi, 1999]: The function $f{:}\alpha \rightarrow \beta$ is:

1. *Order-preserving (i.e. $\uparrow$) :* if $x \leq_\alpha y$, then $f(x) \leq_\beta f(y)$, for every x, y in α.

2. *Order-reversing (i.e. $\downarrow$):* if $x \leq_\alpha y$, then $f(y) \leq_\beta f(x)$, for every x, y in α.

3. *Order-cancelling (i.e. $\not\uparrow\not\downarrow$):* if it is neither 1 nor 2.

Specifying types to the containment relation ($\leq$) and considering f is the context M that a functional type expression creates gets us back to definition 3.1. In English, such expressions (which we will call henceforth *operators*) are mostly monotone (i.e. by default); however, a number of important ones are antitone [MacCartney and Manning, 2007], such as: *negation* (not), some *quantifiers* (every, no, few, etc.), certain *verbs* (refuse, fail, forget, etc.) and *prepositions* (without). It is also important to realize that some operators with multiple arguments have different monotonicity effects on each of their arguments. For instance, '*every*' in (4.1) is antitone on its first argument and monotone on its second, while '*doubt*' is monotone on the first and antitone on the second (see (4.6)). Detecting monotonicity operators and knowing their effect is one step towards establishing the monotonicity property for all positions in a sentence

[Bernardi, 1999; Yanaka et al., 2019]. However, as they might come nested within a sentence, such as '***Every*** *man loves* ***every*** *woman.*' and '*I did* ***not forget*** *to visit him.*', the final marking is computed compositionally in the monotonicity marking part of the calculus.

(4.6)
p. *(He)*$^{\uparrow}$ **doubts that** *(she likes him)* $^{\downarrow}$.
c. He doubts that she loves him.

4.2.2 Monotonicity Marking

Before explaining how marking is done, let us first make a clear distinction between two related features. Consider the examples 4.1, 4.2 and 4.7. In the first two, '*men*' has a different monotonicity due to being an argument of the different operators '*all*' and '*most*', respectively. However, in 4.7(a) and (b), '*men*' is an argument of the same operator '*all*', but, the context it appears in is different; $\downarrow$ in the first and $\uparrow$ in the second, and this is because '*all men*' is itself an argument of another antitone operator '*not*'. That is to say, operators' effects are local to their arguments, while the context property *–polarity–* is a global feature that is computed compositionally for each position in a sentence based on the operator(s) acting on it. Monotonicity marking, therefore, is typically performed on a syntactic analysis of a sentence where the **nesting** of operator-argument(s) construction, such as in 4.7(b), is transparent.

(4.7) a. All *(men)* $^{\downarrow}$ love cars.
b. Not all *(men)*$^{\uparrow}$ love cars.

Syntactical analyses (i.e. parse trees) are structures of sentential elements, called nodes, that are arranged in some kind of hierarchy and are constructed using a grammar. For instance, trees constructed using dependency grammar have a root node, leaf nodes and internal nodes linked in parent–child relations. Given a parse tree, monotonicity marking involves parts, one of which is marking each node with an initial sign of *polarity* (positive (+)(the default), negative (−), and non (#)) based on entries of monotonicity operators (see Table 4.2 for examples and Appendix A for a more comprehensive list), where the aforementioned signs represents the monotonicities $\uparrow$, $\downarrow$, and $\not\uparrow\not\downarrow$respectively. This part could be seen as the *inner marking* step. As for the *outer marking* step, because there are elements with a certain number of arguments (children) and these elements could be within the scope of other elements (i.e. have parents), there must be a type of propagation method that decides on a node's final

polarity, considering all polarities in the path from that node to the root. This propagation process makes use of an operator named the *monotonicity composition operator* ($\circ$), see definition 3.3. To give an example, consider the sentence '*John refused to move without blue jeans*' and its dependency tree in Figure 4.1. Based on the entries of Table 4.2, the sentence has two polarity affecting operators: *refused* $^{\uparrow\downarrow}$ and *without* $^{\downarrow}$. In Figure 4.1, each node on the left tree is marked either with one or two polarities depending on its position in the tree. Nodes in the scope of '*refused*' are marked with blue polarities and others occurring in the scope of '*without*' are marked with green polarities. Nodes with black polarities are set to the default. Now that nodes marked internally, each node with two polarities will apply the monotonicity composition operator on its polarities to get a final one. Once that is done, the resulting tree (Figure 4.1 (right)) reflects the final (outer marking).

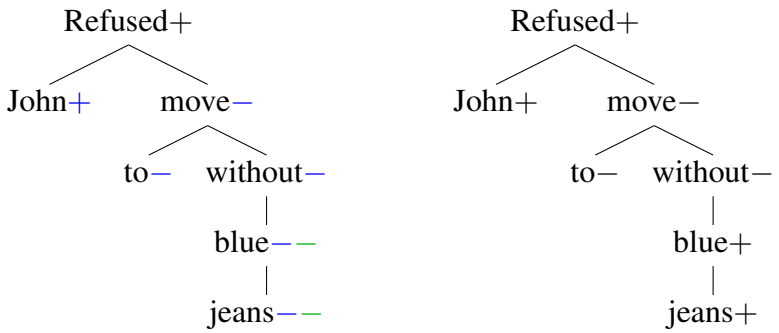


Figure 4.1: An example of inner marking (left) and then outer marking (right).

Monotonicity operator	**First Argument**	**Second Argument**
Every	$\downarrow$	$\uparrow$
Most	~~$\uparrow\downarrow$~~	$\uparrow$
Few	~~$\uparrow\downarrow$~~	$\downarrow$
Without	$\downarrow$	
Refused	$\uparrow$	$\downarrow$

Table 4.2: Example of monotonicity operators.

Definition 3.3: Having a set of polarity markers P={+,−,#} , the monotonicity composition operator $(\circ)$ decides on the final polarity mark as follows [Bernardi, 1999; Icard III and Moss, 2014]:

No polarity (#):	$(\#) \circ (+\vert-)$	=	$(+\vert-) \circ (\#)$	=	$(\#)$
Positive polarity (+):	$(+) \circ (+)$	=	$(-) \circ (-)$	=	$(+)$
Negative polarity (−):	$(+) \circ (-)$	=	$(-) \circ (+)$	=	$(-)$

The description of polarity marking given here is rather general. The reason for not giving discrete steps for the marking process is that most studies in the literature handle it quite differently in several respects. First, the choice of the grammar behind the syntactical derivations will decide the direction in which its nodes are scanned and marked, *top-down* or *bottom-up*. The calculation of final polarity marks could be done in one pass (along with generating the parse tree) or several passes (such as parse, inner mark, and then outer mark). Therefore, concrete examples are given along with their related work in the next section.

4.3 Natural Logic-Based Inference Systems

To date, many NLI systems were based on versions of natural logic; some with novel ingredients (extension to the containment relation, exploring new derivation grammar along with mechanisms to polarity marking, etc), with others incorporating some insight from formal proof systems. In this section we are limiting our review to the **computational** approaches from both kinds, that have been developed in the last two decades (after the first formulation of natural logic as monotonicity calculus) with some reported results on datasets with monotonicity inference problems, particularly FraCas (Section 2.2.1). The reviewed inference systems were grouped (based on the proof method used) into: proof by alignment approaches (Section 4.3.1), order based approaches (Section 4.3.2) and Tableau based approaches (Section 4.3.3). Moreover, all the discussed systems are based on the three-way classification of judgement (Section 2.1). In other words, is the relationship concluded between a premise(s) and a hypothesis *entail* ($p_s \models h$), *contradict* ($p_s \models \neg h$), or *neutral*($p_s \not\models h \wedge (p_s \not\models \neg h)$). Finally, natural logic inference systems are expected to have a grammar to generate syntactical analysis, a mechanism to mark polarities and inference rules along with possibly background knowledge to perform directional replacements. Therefore, among others, the

reviewed systems will be categorized (after being separately discussed) in one table based on these criteria (see Table 4.9).

4.3.1 Proof by Alignment

As mentioned earlier, the monotonicity calculus of Van Benthem(1986) and Sánchez Valencia (1991) was the first attempt to explain inferences about monotonicity as semantic containments and inversions[3] over expressions of different semantic types. However, their calculus was not able to validate several simple inferences, especially those that involve *exclusions*, such as '*Luli is a cat*' $\models$ '*Luli is not a dog*'. Work by Nairn et al. [2006] managed to explain interesting inferences involving implicatives and factives (see Section 5.8.2 and Section 5.8.2 respectively) by defining nine *implication signatures* to identify what implication they have –positive (+), negative (-) or null (0)– in a positive as well as a negative context. For example, '*John refused to dance*'$\models$ '*John did not tango*', because '*refuse*' has a signature -/0 which indicates its negative implication in this positive context. The work of Nairn et al.[2006], nonetheless, was not a direct adaptation to natural logic, thus it lacks containment and inversion relations.

A more comprehensive work that extends the monotonicity calculus to include inferences about exclusion and partially incorporates Narin's signs for implicatives is by [MacCartney and Manning, 2007, 2008, 2009]. Their work is a computational model for a version of natural logic: named *Natlog*. The key parts of the model are: an inventory of *semantic relations* over expression of all types, a replacement of monotonicity with a more general concept *projectivity*, and a light proof procedure of a hypothesis from a premise that involves establishing an *alignment* (i.e. a sequence of *atomic edits*) between the pair, and predict their final entailment relation by *joining* the atomic entailment relations across the edit sequence.

The Inventory of Semantic Relations

To go beyond the semantic containment of monotonicity calculus when modeling the entailment relations of expressions, MacCartney and Manning have introduced seven mutually exclusive entailment relations that were designed by analogy with set relations. Given x and y as the denotation of two expressions (i.e. set of objects

[3] Again, it is just a distinction between containment in upward-entailing context ($\leq$)(Definition 3.1 (1)) and the on in a downward-entailing context ($\geq$)(Definition 3.1 (2)), where inversion is the latter.

in the domain of discourse D that satisfies them), Table 4.3 summarizes these relations along with their set theoretic definition and a demonstration example. Among these relations, although factored into three relations (equivalence ($\equiv$), forward entail ($\sqsubset$)and backward entail ($\sqsupset$)), the semantic containment relation is still preserved[4]. On the other hand, semantic exclusion is represented with the ($\curlywedge$ and $\Downarrow\!\!\!\uparrow$), and indirectly ($\smile$)[MacCartney and Manning, 2009].

Symbol	Name	Example	Set theoretic definition
$x \equiv y$	equivalence	couch $\equiv$ sofa	$x = y$
x $\sqsubset$ y	forward entailment	crow $\sqsubset$ bird	$x \subset y$
x $\sqsupset$ y	reverse entailment	European$\sqsupset$ French	$x \supset y$
x $\curlywedge$ y	negation	human $\curlywedge$ nonhuman	$x \cap y = \phi \wedge x \cup y = D$
x $\Downarrow\!\!\!\uparrow$ y	alternation	cat $\Downarrow\!\!\!\uparrow$dog	$x \cap y = \phi \wedge x \cup y \neq D$
x $\smile$ y	cover	animal $\smile$ nonhuman	$x \cap y \neq \phi \wedge x \cup y = D$
x # y	independence	hungry # hippo	(all other cases)

Table 4.3: The basic semantic relations of the NatLog system [MacCartney and Manning, 2009].

Projectivity

The examples shown above (in Table 4.3) illustrate entailment relations at the level of lexical items, hence are called *lexical entailment relations*. The question is how these lexical relations are *projected* in their containing compound expression (sentences for simplicity). As seen in Definition 3.2 of the monotonicity calculus, the monotonicity of an operator, $\uparrow$, $\downarrow$ or $\not\uparrow\not\downarrow$, determines whether the containment relation between two expressions is preserved (projected without change), reversed or turned into none respectively, when appearing in the context of that operator. As an example, we have seen in 4.3, when applying the operator (no $^{\downarrow}\,^{\downarrow}$), the relation '*red car*' $\leq$'*car*' is reversed i.e. '*no car*' $\leq$'*no red car*'. Similarly, MacCartney and Manning assigned for each operator a *projectivity signature*. These signatures are an extension to the three classes of monotonicity and the nine implicative signs of Nairn et al., such that, given β, the set of entailment relations, $\{\equiv, \sqsubset, \sqsupset, \curlywedge, \Downarrow\!\!\!\uparrow, \smile, \#\}$, each signature is defined as a

[4] MacCartney and Manning's [2007] use of different symbols for semantic containment relation was motivated by their set theoretic definition; considering that it is parallel in meaning to the containment relation in set theory ($\subseteq$). However, that choice does not imply having a different meaning to the semantic containment relation defined earlier with respect to ($\leq$).

mapping $m : \beta \rightarrow \beta$. Taking the negation word '*not*' for example, it will project the relations ($\equiv$, $\curlywedge$ and #) without change and swap between ($\sqsubset$ and $\sqsupset$) and ($\Downarrow$ and $\smile$) and hence its projectivity signature is as defined in Table 4.4 [MacCartney and Manning, 2008]. Ideally, projectivity is computed and marked on a semantic composition tree to ensure the right nesting of function-argument(s) constructions. However, MacCartney and Manning rely on a phrase-structure tree produced using the *Stanford statistical parser* [Klein and Manning, 2003]. The choice of phrase-structure trees has complicated the process of marking projectivity as the nesting of constituents does not always comply with their ideal composition in a semantic tree [MacCartney and Manning, 2008].

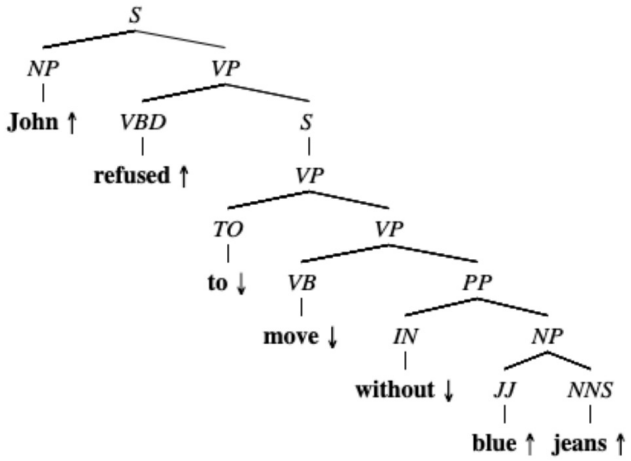


Figure 4.2: An example of a monotonicity marked phrase-structure tree.

For example, in the parse tree of '*John refused to move without blue jeans*'(Figure 4.2)[5] the operator '*without*', is not the **parent** of its argument '*blue jeans*', i.e. the argument is not in its parent scope. To amend this shortcoming, they have employed predefined patterns called *Tregex patterns* in which each projectivity affecting operator will be defined in terms of its arity, the syntactic pattern that identifies its occurrence in the tree, and the affected arguments. Each argument has a projectivity sign[6] and

[5]The actual parse of the sentence was from [MacCartney, 2009] with a slight change, while its graphical representation was drawn using an online tool (http://mshang.ca/syntree/) on which I have manually added the monotonicity marks for illustration.

[6]MacCartney has illustrated in his thesis [2009] the projectivity effect of several operators including (logical connectives, quantifiers, verbs and implicatives) as a map from β to β. Nonetheless, in the actual implementation of the model, NatLog, only monotonicity signs have been computed and marked on trees. That is, I believe, why arguments in the Tregex patterns have only $\uparrow$, $\downarrow$ or $\not\updownarrow$as their projectivity signs. Practically, antitone operators will have the same projectivity signature as the one defined in Table 4.4; monotone operators are identity functions project without change and non-monotone will project any relation to #.

a Tregex pattern of its own that determines the affected span of this projectivity sign. For example, in Figure 4.3, *'without'* has an antitone effect on one argument of the type propositional phrase (PP) which appears directly on its right. These patterns, along with the monotonicity composition function (Definition 3.3) will be used in a bottom-up scan of phrase-structure trees to mark them with final projectivity signs.

r	$\equiv$	$\sqsubset$	$\sqsupset$	$\Updownarrow$	$\smile$	$\wedge$	#
$m(r)$	$\equiv$	$\sqsupset$	$\sqsubset$	$\smile$	$\Updownarrow$	$\wedge$	#

Table 4.4: The projectivity signature for '*not*', in which r is a relation and $m(r)$ is the projected relation [Angeli and Manning, 2014]

unary operator: *without*
  pattern: `IN < /^[Ww]ithout\$/`
  argument 1: monotonicity ↓ on dominating PP
    pattern: `___ > PP=proj`

binary operator: *most*
  pattern: `JJS < /^[Mm]ost\$/ !> QP`
  argument 1: monotonicity ⇅ on dominating NP
    pattern: `___ >+(NP) (NP=proj !> NP)`
  argument 2: monotonicity ↑ on dominating S
    pattern: `___ >+(/.*/) (S=proj !> S)`

Figure 4.3: Example of Tregex patterns for *'without'* and *'most'* [MacCartney and Manning, 2007].

Proof by Alignment

MacCartney and Manning's [2007] procedure for proving a hypothesis h from a premise p proceeds as incremental atomic edits (*delete (DEL), insert (INS), substitute (SUB) and advance (ADV)* i.e pass without change), that links the pair, i.e. alignment. Each atomic edit will induce[7] a lexical entailment relation, which in turn is projected to the sentence level into an *atomic entailment relation* between a sentence and its mutated (by that atomic edit) version. Finally, these atomic entailment relations are iteratively *joined*($\bowtie$), according to Figure 4.4, into a global entailment relation between p and h.

To demonstrate, consider the example in Table 4.5, in which S_0 and S_5 are the premise and the hypothesis respectively. The sequence of edits are $e_1 - e_5$. Each edit

[7]Using a classifier that predicts a relation from information about the lexical items involved in the edit and the kind of edit that has been applied (deploying measures from WordNet and Levenshtein edit algorithm), see [MacCartney, 2009] for more details.

(e_i) will introduce a lexical relation (r_i), which then is projected into an atomic relation $m(r_i)$ between the sentences $S_{i\text{-}1}$ and S_i, according to the projectivity signature of the containing context. Take the edit e_3: SUB(move, dance) for example. The lexical relation between '*move*' and its substituent '*dance*' is ($\sqsupset$), and it is projected as ($\sqsubset$) between S_2 and S_3, as both lexical items appear in the context of the negation '*didn't*' which, as defined in Table 4.4, swaps between ($\sqsubset$, $\sqsupset$). The final entailment relation between S_0 and S_5 is the one at E_5 in Table 4.5 (i.e. $\sqsubset$) and is iteratively determined, with E_0 axiomatically initiated to $\equiv$, as : $E_i := E_{i\text{-}1} \bowtie m(r_i)$ [Angeli and Manning, 2014].

i	S_i	Edit (e_i)	r_i	$m(r_i)$	E_i
0	*John refused to move without blue jeans*				
1	*John moved without blue jeans*	DEL(refused to)	⇵	⇵	⇵
2	*John didn't move without blue jeans*	INS(didn't)	⅄	⅄	$\sqsubset$
3	*John didn't dance without blue jeans*	SUB(move, dance)	$\sqsupset$	$\sqsubset$	$\sqsubset$
4	*John didn't dance without jeans*	DEL(blue)	$\sqsubset$	$\sqsubset$	$\sqsubset$
5	*John didn't dance without pants*	SUB(jeans, pants)	$\sqsubset$	$\sqsubset$	$\sqsubset$

Table 4.5: An example of MacCartney and Manning proof procedure for a hypothesis S_5 from the premise S_0 [Abzianidze, 2017b].

$\bowtie$	$\equiv$	$\sqsubseteq$	$\sqsupseteq$	⅄	⇵	$\smile$	#
$\equiv$	$\equiv$	$\sqsubseteq$	$\sqsupseteq$	⅄	⇵	$\smile$	#
$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	#	↾	⇵	#	#
$\sqsupseteq$	$\sqsupseteq$	#	$\sqsupseteq$	$\smile$	#	$\smile$	#
⅄	⅄	$\smile$	↿	$\equiv$	$\sqsupseteq$	$\sqsubseteq$	#
⇵	⇵	#	↿	$\sqsubseteq$	#	$\sqsubseteq$	#
$\smile$	$\smile$	$\smile$	#	$\sqsupseteq$	$\sqsupseteq$	#	#
#	#	#	#	#	#	#	#

Figure 4.4: The join table in which each entry is a result of joining ($\bowtie$) two entailment relations (i.e. a row with a column) [Icard III, 2012; Angeli and Manning, 2014].

In conclusion, referring to the correspondences in (Table 4.6), their model of natural logic handles single-premise inferences on section (1, 2, 5, 6 and 9) of the FraCaS dataset with an average accuracy of 87%, and 59% accuracy on the third *recognizing textual entailment* RTE3 challenge [Giampiccolo et al., 2007] by its own, however when used as a component in another RTE system [Chambers et al., 2007] the overall accuracy went up to 63%. The simplicity of its approach is appealing, but the NatLog system is significantly crippled by the usage of alignment as proof procedure [Abzianidze, 2017b]. First, it is possible to have several alignments linking a $(p-h)$ pair of

which not all share the same final entailment relation. Therefore, there is a need to find the alignment that leads to the **correct** final entailment between a $(p-h)$ pair. Moreover, the procedure is sensitive to word order, thus, it cannot sanction inferences such '*John wrote a book.*' $\models$ '*A book was written by John.*'[Lewis and Steedman, 2013]. Second, bound to a pair of expressions, the alignment-based approach falls short when it comes to reasoning over multi-premises. Finally, MacCartney and Manning [2008, 2009] observed that their model for natural logic has less deductive power than FOL as it can not account for fairly simple inferences including de Morgan's laws for quantifiers (e.g '*not all birds fly.*' $\models$ '*some birds do not fly*') [Abzianidze, 2017b; MacCartney and Manning, 2008].

Relation	It symbol	Judgement
Equivalence	$p \equiv h$	entails
Forward entailment	$p \sqsubset h$	entails
Alternation	$p \Downarrow h$	contradicts
Reverse entailment	$p \sqsupset h$	neutral
Independence	$p \# h$	neutral

Table 4.6: The correspondence between the NatLog final entailment relation between p and h and the 3-way judgement for datasets [MacCartney and Manning, 2007].

4.3.2 Order-Based Approach

As mentioned in Section 4.2, in the model-theoretic semantics of NL it is assumed that denotations of linguistic expressions are depicted as objects in domains which are partially ordered. That is to say, the meaning of expressions (of the same type) are comparable [Fyodorov et al., 2003] and hence maintain some semantic order-relations ($\leq$). Moreover, according to Frege's principle of semantic composition, order relations between compound expressions are derived from orders between their constituents, according to the rules of the deriving grammar and the semantic properties of operators [Fyodorov et al., 2003]. One important example of such properties is monotonicity and its effect on order-relations is given in Definition 3.2.

Order-based approaches are based on the view that in an adequate semantic theory, order-relations (orders for short), such as in 4.2 (see Section 4.2.1 for its discussion), correspond to an intuitively valid entailment relation [Fyodorov et al., 2003; Zamansky et al., 2006]. Therefore, based on insights from model theory but with no direct appeal to models, a few computational models were built contributing to the natural logic

endeavour as semantically annotated syntactic derivations of NL expressions are used as a basis for computing inferences.

This approach started with the computational model of Fyodorov et al. [2003] and Zamansky et al. [2006] for a fragment of English. Nonetheless, its most recent descendent, the work of Hu et al. [2019a] and Hu et al. [2019b] is the one that targeted inferences of the kind appearing in the FraCas and SICK datasets. Generally, each of these computational systems relies on a version of *Combinatory Categorical Grammar* (CCG) parsers to produce the syntactical analysis of NL expressions. Then nodes of derivation trees are decorated with some semantic features (such as monotonicity and restrictive modifiers (Section 5.4)). After that, an entailment relation between a premise p_s and a hypothesis h, is determined mainly by computing from p_s an inference or a contradiction that matches h. The judgement is entail if h matches an inference from p_s, contradict if the match is with a contradiction, and neutral otherwise. Inferences and contradictions are mainly derived by the iterative *replacement* of a constituent from p_s with another of the same type making use of existing orders (see below) in a way that the decorated features of the constituent are obeyed. There are, possibly, some other rules to handle inferences that cannot be tackled using replacement, depending on the aim and the scope of the work. Orders used in replacements are mainly from three sources [Fyodorov et al., 2003]:

- *Construction-based*: which concerns orders coming from specific linguistic constructions in a language such as : *red car* $\leq$*car*, *very red* $\leq$*red*, and *very red car* $\leq$*red car* $\leq$*car*.
- *Lexical-based*: which are orders appearing between words due to their meaning such as: *dog*$\leq$*animal* and *walk*$\leq$*move*.
- *Imported*: from external sources or inferred during a proof process.

To give a demonstration of these parts, consider the inference system of Hu et al. [2019a] and Hu et al. [2019b] named *MonaLog*. It focuses on reasoning about monotonicity. Therefore, it takes as input an argument of which the premises are decorated with monotonicity signs (e.g. $most^{\uparrow}Europeans \not\equiv live^{\uparrow}outside \not\equiv of \not\equiv Asia^{\downarrow}$)[8]. The system maintains two sets: knowledge-base (K) and sentence-base (S). The first includes orders which are from the above-mentioned three sources. The initial fixed K contained lexical relations that are imported from WordNet (e.g. $dog \leq animal$, dog

[8]We replaced their ($^{=}$) with ($\not\equiv$) for readability.

| *cat*) in addition to some other hand-coded orders about quantifiers (e.g *every* = *all* = *each* $\leq$ *most* $\leq$ *many* $\leq$ *a few* = *several* $\leq$ *some* = *a*)[Hu et al., 2019a]. The set K also includes orders that are related to the inference problem in hand; orders that are of specific linguistic construction appearing in the set of premises p_s or h. For instance, for each verb v and adverb a, in p_s or h, the order $a\ v \leq v$ is added to K. The second set S includes every inference and contradiction that can be derived by the system starting from p_s by iteratively performing two types of replacements on each p [Hu et al., 2019a]:

- `replacement_infer`: replaces a constituent with something bigger or smaller in K depending on the mark on constituent, whether it is $\uparrow$ or $\downarrow$, respectively. Then it adds the inferred the sentence to $S.inferences$.
- `replacement_contra`: Negates the sentence by adding, for example, '*do not*' before the main verb, or by replacing the quantifier '*some*' with '*no*' and vice versa. Then it adds the inferred sentence to $S.contradiction$.

Note that `replacement` does derive many inferences, but not all of them [Hu et al., 2018]. Hence, the system incorporates other natural logic rules, such as those in Table 4.7. The successful application of these rules, which requires minimal editing[9], will add new sentences to the list S.

Rule	$\dfrac{Some\ y\ are\ x}{Some\ x\ are\ y}\ SOME_2$	$\dfrac{Det\ x\ y \quad All\ x\ z}{Det\ x\ (y \wedge z)}\ DET$
Example	$\dfrac{Some\ cats\ are\ animals}{Some\ animals\ are\ cats}$	$\dfrac{Every\ cat\ is\ an\ animal \quad All\ cats\ meow}{Every\ cat\ is\ an\ animal\ who\ meows}$

Table 4.7: Examples of Hu et al.'s rules of natural logic [Hu et al., 2019a].

The updated list S is then searched using a *depth-first search* [10] algorithm (with depth=2) to find string-for-string match of h. If a match is found in $S.inferences$ before the stopping criteria is reached, the returned judgement will be entail. If not, $S.contradictions$ is then searched and, if a match is found, contradict will be returned, otherwise unknown is returned. Figure 4.5 illustrates as a search tree, the inferences and contradictions derived from a premise '*every animal likes some young semanticist*'

[9] Hu et al. stated that before rules a application, NL sentences are turned into compatible format, then convert the resulted inference back to a NL sentence (e.g from *every cat (animal* $\wedge$ *meow)* into *every cat is an animal who meows*).

[10] It is an AI search algorithm that transverses a tree data structure starting from its root and exploring as far as possible along each branch before backtracking [Wikipedia contributors, 2020].

in an attempt to prove the hypothesis '*every cat likes some linguist*'. At the root lies the marked premise followed by inferences on the branches, each branch corresponds to a single replacement (e.g. the left-most branch is the result from replacing *every* with *some*). Then, each inference on these branches will have its own inferences and contradiction and so on. Searching the tree results in an entail relation as the hypothesis matched the inference appearing as a leaf in the right-most branch of the search tree.

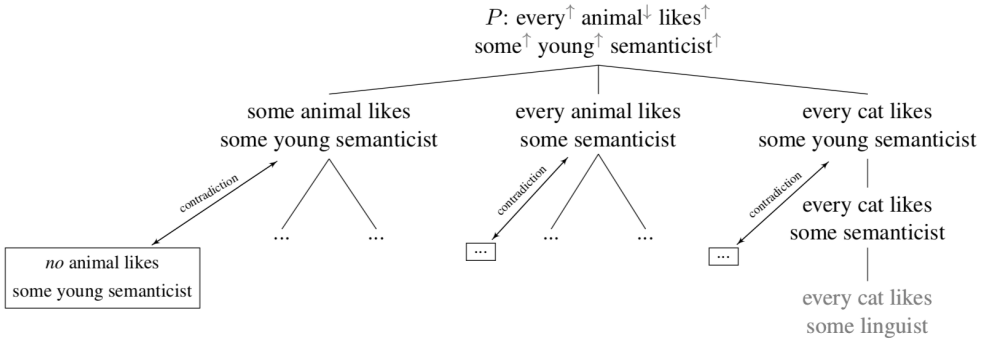


Figure 4.5: Search tree example that starts with the premise '*every animal like some young semanticist*' [Hu et al., 2019a].

A final remark on Hu et al. [2019a] and Hu et al. [2019b] system is that it relies on the *polarizing tool*[11] of Hu et al. [2018] for marking p_s with monotonicity. The key features of this tool are: the use of (1) a mapping mechanism from CCG syntactic categories to semantic types; (2) a lexicon of expressions with *polarity-enriched* semantic types, and some rules to handle them. The tool does the marking in two phases [Hu and Moss, 2018]. First, it scans the tree from leaves to root and assigns polarities to types according to the lexicon entries and the associated rules. Then, it scans the tree again from root to leaves to compositionally determine monotonicity signs of words. Figure 4.6 shows a simple example of such marking, in which (a) is the CCG derivation of '*Fido chased Felix*' and (b) its polarity-marked version. The final output ($Fido^{\uparrow} chased^{\uparrow} Felix^{\uparrow}$) is read from the leaves of the marked tree. The tool accepts CCG parses either hand-created parses or ones derived using a popular and freely available CCG parser [Hu and Moss, 2018]. That allowed Hu et al. [2019a] to experiment with two different parsers, the *C&C* parser of Curran et al. [2007] and the *Easy CCG* parser of Lewis and Steedman [2014], and perform some modification on trees, either to obtain semantically more meaningful trees, or to correct parsing errors and inconsistencies.

[11] https://github.com/huhailinguist/ccg2mono

$$\dfrac{\dfrac{\dfrac{\textit{Fido} : n_{pr}}{\textit{Fido: } s/(s\backslash n_{pr})}\,\mathrm{T} \qquad \textit{ch: } (s\backslash n_{pr})/n_{pr}}{\textit{Fido chased} : s/n_{pr}}\,\mathrm{B} \qquad \textit{Felix} : n_{pr}}{\textit{Fido chased Felix} : s}\,>$$

(a)

$$\dfrac{\dfrac{\dfrac{\textit{Fido}^{\uparrow} : e}{\textit{Fido}^{\uparrow} : et \xrightarrow{+} t}\,\mathrm{T} \qquad \textit{chased}^{\uparrow} : e \xrightarrow{+} et}{\textit{Fido chased}^{\uparrow} : e \xrightarrow{+} t}\,\mathrm{B} \qquad \textit{Felix}^{\uparrow} : e}{\textit{Fido chased Felix}^{\uparrow} : t}\,>$$

(b)

Figure 4.6: Example of a CCG tree before and after monotonicity marking using the tool of Hu et al. [2018].

The work of Hu et al. [2019a] is limited to only the first section of the FraCas data set (i.e. monotonicity examples) that is handled with 88 % accuracy. Nonetheless, it claims a trivial extendibility to other sections by augmenting the list of inference rules. Moreover, the [Hu et al., 2019b] version of MonaLog, reported 77.19 % accuracy on the test part of the *SICK dataset* [Marelli et al., 2014]. In comparison with the alignment procedure of the NatLog system, the MonaLog proof procedure avoids the need for expensive search for the sequence of edits leading to a correct inference, as replacements can happen in any order order, each of which are a step towards finding a proof. Hu et al. also claim to have a simpler proof procedure and MRs than the natural tableau system (Section 4.3.3).

4.3.3 Natural Tableau

This is a tableau theorem prover for natural logic; a novel formal method for reasoning over representations that are linguistically relevant [Abzianidze, 2014]. The theory behind that proof system was first introduced by [Muskens, 2010] and gradually extended to a wide-coverage automated theorem prover, namely *LangPro*, by Abzianidze in [2014; 2015; 2016; 2017a].

Muskens's proof method is based on a signed version of the analytical tableau of D'Agostino and Mondadori [1994] that is fed with a representation of a NL argument called *Lambda Logical Forms (LLF$_s$)*. LLF$_s$, as in (4.8)[12], are simply typed lambda

[12]These examples were taken from the original article [Muskens, 2010]. Although they were supposed to be examples of **typed** λ-terms, Muskens chose to omit type information in his illustration of examples.

terms that are built, via functional application and lambda abstraction, from variables and non-logical constants (i.e. lexical terms) only, in order to maintain a close relevance to their linguistic surface forms (which is a characteristic of *natural* logic).

(4.8) a. `((a woman) walk)`

b. `((if((a woman)walk)))((no man)talk))`

c. `(mary(think((if((a woman))walk))((no man)talk)))`

d. `(few man)λx.(most woman)λy.like xy`

Natural tableau, as any tableau system[13], is a refutation system. Its proof procedure starts with the counterexample (i.e. the premises are true and the conclusion is false) for the argument to be proven and follow it with a series of inference rules application until a closed tableau is reached (i.e. all branches are closed and marked with $\times$), hence the original argument is proved, or there are no more rules to be applied. However, while tableau systems typically have a handful of inference rules, natural tableau was designed with an inventory of many rules, rules that are connected to specific classes of expressions [Muskens, 2010]. Muskens's rules were not aimed for wide-coverage; as his focus was to provide rules that can be argued to come close to the rules of human reasoning. Thus, they were limited to a group of interesting semantic phenomena including: monotonicity, anti-additivity and determiners in addition to some classical boolean rules about logical constants, closure rules and rules derived from the format of the natural tableau entries (see Table 4.8).

A natural tableau entry is either $T\vec{C}:A$ or $F\vec{C}:A$, in which T and F are truth signs, A is a LLF of type $\langle\vec{\alpha}\rangle$[14] and $\vec{C}$ represents an argument list of constants or LLF_s of type $\vec{\alpha}$ [Muskens, 2010; Abzianidze, 2015]. As for its semantics, an entry is intuitively stating that the application of A to its list of arguments $\vec{C}$ is evaluated to its truth sign [Muskens, 2010]. For example, Figure 4.7, illustrates a successful (i.e. closed) tableau proof of '*no lark flew*' from '*no bird moved*'. In a given world i, the proof's entries are: the true premise (T_i : `(no bird) moved`) and the false consequence (F_i : `(no lark) flew`), while the applied rules are the ones developed for antitone operators (given in Table 4.8(b)) due to the quantifier '*no*', plus the closure rules. Muskens's theory also assumes order-relations at the level of lexical items, between words (such as `lark`

[13] See [Fitting, 1990, ch 3 & 6] an elaborated explanation of propositional and FOL tableau systems.

[14] Muskens specified the following conventions for types: (1) they are recursively built from the primitives: t (truth value), e (entity) and s (states); (2) they are relational i.e. $\langle\alpha\rangle$ type is the functional type $\alpha \to t$ and $\langle\rangle$ is equivalent to t; and (3) they are left associative [Muskens, 2010].

$\leq$ bird and no $\leq$ few [Muskens, 2010]) to be given in the lexicon knowledge of the system. This relation explains the second closure rule in Table 4.8), which in turn licences the closure of the two left-most branches in Figure 4.7. A reader of Muskens's work may notice, in contrast to the other approaches that have been discussed so far, there exists no explicit mechanism for marking monotonicity. However, we believe that entries signs act like one and they flip, when nested as an argument of a monotonicity changing operator, accordingly. For example, Figure 4.8 shows that the argument of '*every*' is antitone in (a) and hence we can infer '*every man*' from '*every human*'. However, it is monotone in (b) because '*every*' is an argument of '*not*' which, as its rules in Table 4.8 shows, flips the truth sign (i.e. the monotonicity) of its argument and therefore '*not every human*' is the one inferred from '*not every man*'.

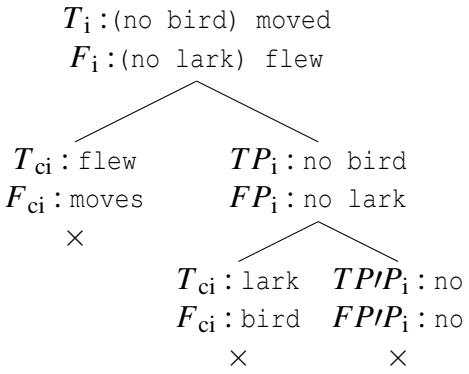


Figure 4.7: A proof example using natural tableau.[Muskens, 2010].

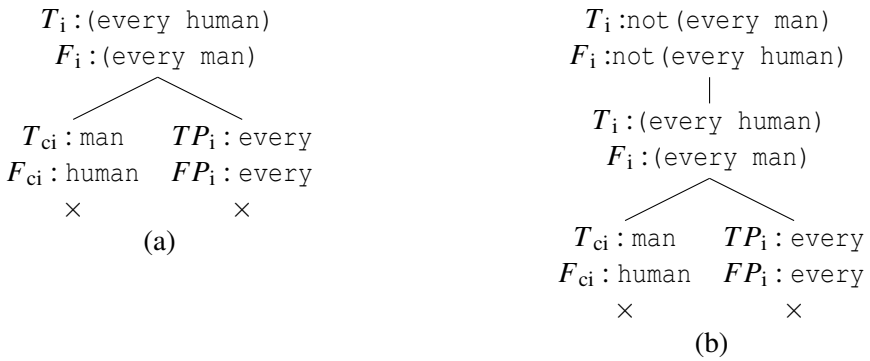


Figure 4.8: Example of indirect account for monotonicity marking in natural tableau.

In general, the proof system, as we mentioned earlier, is a novel approach that adapted a traditional proof theory with formulas that are to a large extent, linguistically relevant. The system bears higher order properties due to the employed simple type theory. The nature of its proofs is transparent and deductive, hence it is capable of

Closure rules:	$T\vec{C} : A$ $F\vec{C} : A$ | $\times$	$T\vec{C} : A$ $F\vec{C} : B$ | $\times$ Where A $\leq$ B
Monotonicity rules:	$T\vec{C}_{\text{i}}$:GA $F\vec{C}_{\text{i}}$:HB $T\vec{c}_{\text{i}}$:A $\quad TbC_{\text{i}}$:G $F\vec{c}_{\text{i}}$:B $\quad FbC_{\text{i}}$:H (a) Tableau rule for monotone operators ($\uparrow$), in which G or H are($\uparrow$) and $\vec{c}$ and b are fresh	$T\vec{C}_{\text{i}}$:GA $F\vec{C}_{\text{i}}$:HB $T\vec{c}_{\text{i}}$:B $\quad TbC_{\text{i}}$:G $F\vec{c}_{\text{i}}$:A $\quad FbC_{\text{i}}$:H (b) Tableau rule for antitone operators ($\downarrow$), in which G or H are($\downarrow$) and $\vec{c}$ and b are fresh
Rules of `not`:	$T\vec{C}$: notA | $F\vec{C} : A$	$F\vec{C}$: notA | $T\vec{C} : A$
Argument shifting rules:	$X\vec{C} : A\mathbf{B}$ | $X\mathbf{B}\vec{C} : A$ Shift to the back	$X\mathbf{B}\vec{C} : A$ | $X\vec{C} : A\mathbf{B}$ Shift to the front

Table 4.8: Examples of Muskens's [2010] tableau rules.

solving arguments with multi-premises [Abzianidze, 2017a]. Furthermore, Muskens argues that a proof-theory, such as the one presented in his work, should complement traditional model-theoretic methods used in the computational study of NL semantics [Muskens, 2010]. Despite the insightful possibilities provided in Muskens' system, no practical experiments were conducted until the implementation of LangPro [Abzianidze, 2014].

LangPro is a wide-coverage automated theorem prover for NL that was developed by Abzianidze in [2014; 2015; 2016; 2017a]. As shown in Figure 4.9, LangPro will take a NL argument, of premises ($p_1 \dots p_{\text{n}}$) and a hypothesis (h), as input and output a judgement. To do so, LangPro goes through phases. First, the input texts are parsed using parsers for CCG. For experimental reasons, two different CCG parsers

were **separately**[15] employed: C&C and Easy CCG parsers. Then, trees are turned automatically into LLFs using the *LLFgen*. For multiple reasons, one of which is that the CCG sometimes produces incorrect analyses, LLFs are not obtained directly from trees; they go through several filters and transformational procedures, as depicted in Figure 4.10 and explained by Abzianidze [2014]. A brief illustration of these steps is given in Figure 4.12, in which (a) is the C&C parse for '*there is no one cutting a tomato*', (b) is the fixed CCG term obtained from the parse tree after the removal of directionality and analysis corrections and (c) shows in 2 and 3 the λ terms that were obtained from the CCG term in 1. These λ terms are the LLFs which are then used in the natural logic tableau prover: *NLog Prover*.

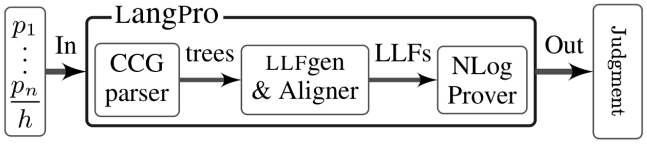


Figure 4.9: LangPro architecture [Abzianidze, 2017a].

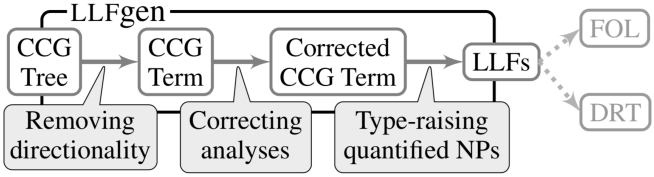


Figure 4.10: LLFgen architecture [Abzianidze, 2017a].

NLog prover is a Prolog implementation of an extended version of Muskens's natural tableau. The extension is three-fold and addresses the typing system, the prover components and a slight reformation to the tableau entries. The typing of the LLFs' lexical items is based on syntactical atomic types $\{s, np, n, pp\}$, which are motivated by CCG syntactic categories. One obvious way to integrate these LLF$_s$ with such typing in Muskens's natural tableau is to map them into semantic types that are composed of the $\{t, e\}$ primitives similar to what is done in the polarizing tool of Hu et al. [2018] and Hu and Moss [2018]. However, in order to facilitate a fine-grained matching during rules application, Abzianidze chose to retain syntactic types in addition to the semantic

[15] Abzianidze's study, in its experiment and evaluation phase used two versions of the prover. One takes its CCG derivations from the C&C parser and the other from the EasyCCG parser. The results were compared on the evaluation dataset to a hybrid prover that considers an answer from either of the two provers as a proof for the problem in hand.

types and establish *subtyping* relations (defined as a partial order over types) between the two typing systems in addition to some typing rules. For example, given (*e* subtype *np*) and the rule that says: if (α subtype β) and a term is of type α , then it is also of type β, applying the term ($\texttt{love}_{\texttt{np,np,s}}\ \texttt{mary}_{\texttt{np}}$) to the constant ($\texttt{c}_{\texttt{e}}$) during a proof is straightforward and needs no mapping [Abzianidze, 2014].

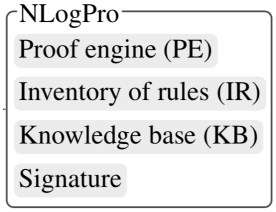


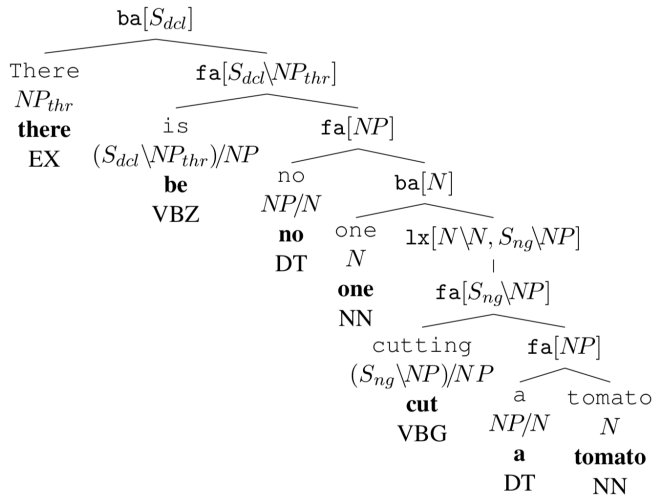
Figure 4.11: NLogPro components [Abzianidze, 2016].

As for the prover components, shown in Figure 4.11, the *signature* lists entries of lexical items that carry some algebraic properties (such as monotonicity and implicativity) that are deemed relevant for inference [Abzianidze, 2017a]. For example, '*every*' has the entry `[dw,up]` specifying an antitone property for its first argument and a monotone for its second. The second part of the prover is the knowledge base and it includes only lexical relations of type hyponym, similarity and antonymy provided in WordNet 3.0 [Fellbaum, 1998]. The inventory of rules, the third part, includes all of the tableau rules used by the NLog prover, and the number of these rules has reached 80 in the system's latest version [Abzianidze, 2017a]. Muskens's essential rules were about 20 and rest were all Abzianidze's extensions. In the 2014; 2015 versions of LangPro, about 30 rules were manually designed and added to the inventory via learning, while the prover attempts to solve the trial part of the SICK dataset (about 500 $p-h$ pairs). The rest of the rules were added by Abzianidze [2016, 2017a], to model deep semantic phenomena and multi-premise problems appearing in the FraCas data set. Generally, the new rules cover both syntactically and semantically motivated phenomena, to name a few: prepositional phrases, passive constructions, auxiliaries, modifiers, copulas, and expletive sentences [Abzianidze, 2015]. Note that learning datasets not only extended the rules but also the signature entries, the knowledge base with lexical relations (such as woman $\leq$ lady) that were not available in WordNet and the LLFgen with more fixing procedures.The final part of the prover is the proof engine, which builds tableau proofs for a given argument making use of the above components. Figure 4.13 illustrates a proof of '*some bird does not fly*' from '*not all birds fly*' and lists the tableau rules used. It can be noticed from the example that the reformation of tableau entries takes on the

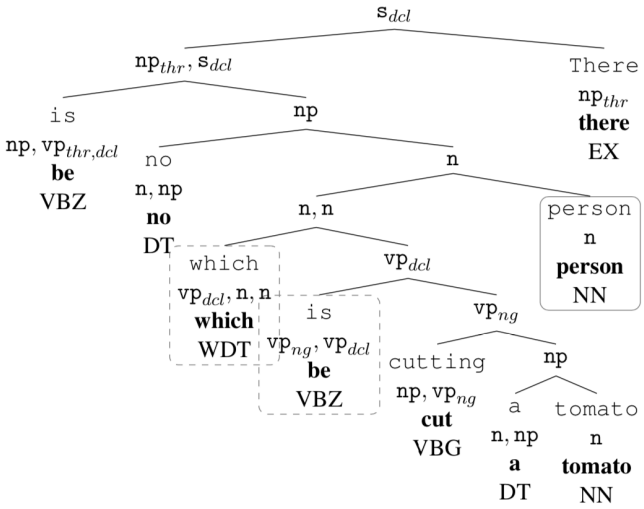
shape: *LLF : argument list: truth sign* [Abzianidze, 2014]. For illustrative purposes, a fourth part on the left of *LLF* includes a numbering for entries and the name of the rule and the number of the argument that it was applied to, resulting in that entry. For instance, $5\ ^{\text{NOT[4]}}\texttt{all[bird,fly]:T}$, is the tableau entry number 5 that has resulted from applying the rule of NOT on entry number 4.

All in all, LangPro have shown state-of-the-art competence with 82.1% accuracy on the test part of the SICK dataset and 87% on certain sections (1, 2, 5 and 9) of the FraCas dataset. Moreover, LangPro[16] is available for those interested in experimenting with the prover. In addition, their LLFs "encode instructions for semantic composition, and hence they can be used to compositionally derive semantics in other meaning representations" [Abzianidze, 2017a] (e.g. FOL and Discourse Representation Theory (DRT)). On the other hand, LangPro was designed to not account for tense or aspect (following the RTE guidelines). Therefore, their system will wrongly derive several trivial yet important inferences involving time such as '*John* ***loved*** *Mary*'$\models$ '*John* ***loves*** *Mary*'. Moreover, there is no special treatment for default sentences and their treatment for attitudes was limited to implicatives and factives.

[16] https://github.com/kovvalsky/langpro



(a) A CCG tree in which each node is labelled with: a token, CCG category, lemma and part-of-speech tag.



(b)

1. `be(no(which(be(cut(a tomato)))person)) there`
2. `no(which(be(`λx`.a tomato(`λy`. cut` yx`)))person)(`λz`. be` x `there)`
3. `a tomato (`λx`. no(which(be(cut` x`))person)(`λy`. be` z `there))`

(c)

Figure 4.12: From CCG tree to LLF_s example [Abzianidze, 2015].

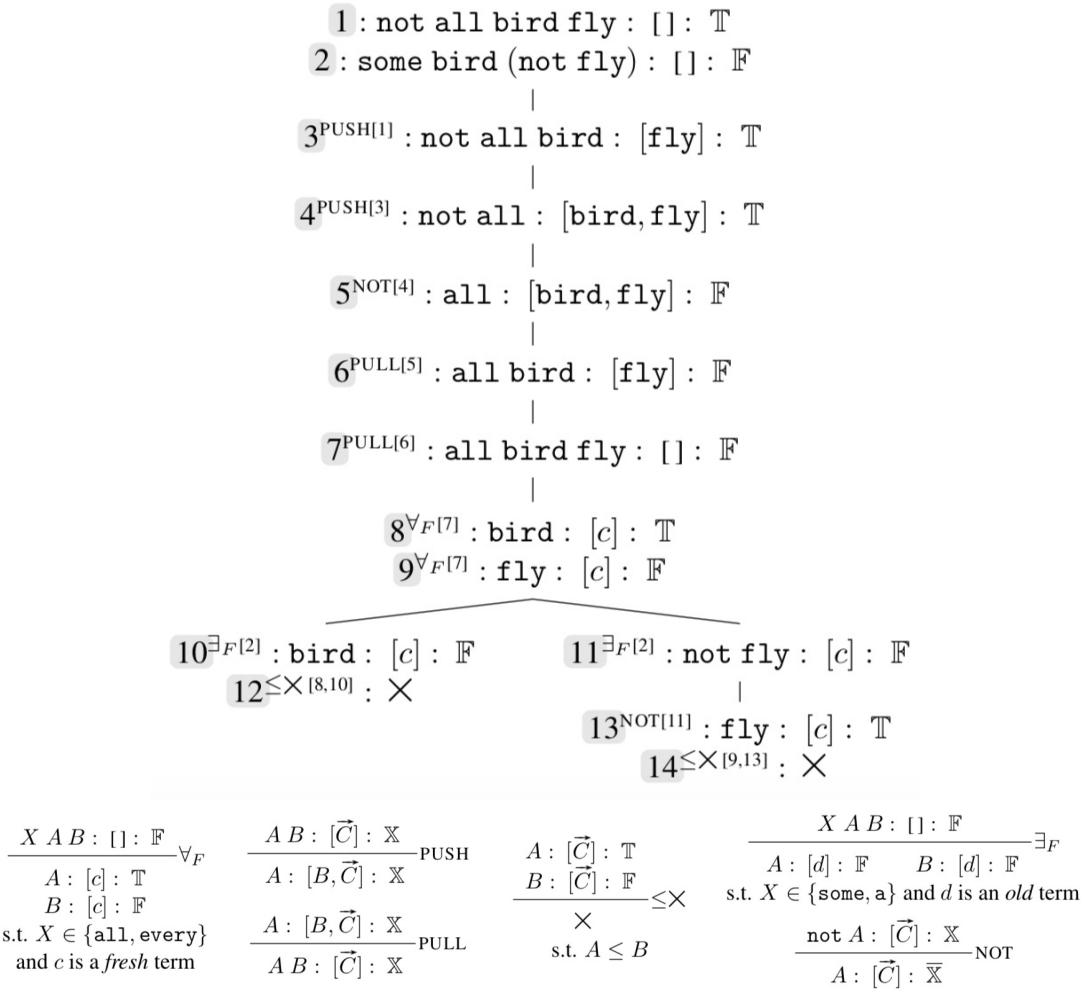


Figure 4.13: A tableau (top) for '*not all birds fly*'$\models$ '*some bird does not fly*' and the list of applied rules (bottom) [Abzianidze, 2015].

4.4 Summary

The aim of this work is to investigate the use of syntactical trees as a basis for reasoning. We also aim to define a matching algorithm on trees (CO3) to be used in our NLI system. The discussion in this chapter was split into two parts. The first was to gain a better understanding of the concepts that contribute to the development of our matcher (Section 8.4), including natural logic (Section 4.1) and one of its most common features: monotonicity (Section 4.2) along with its semantic (containment relation) and syntactic (polarity marking) aspects.

The second part of this chapter has surveyed several natural-logic based inference systems and summarised them in Table 4.9. While reviewing these systems we have particularly focused on certain pints:

1. the ability to do multi-premises inferences.
2. the use of a proof search mechanism to do so.
3. the ability to systematically handle certain semantic phenomena (generalized quantifiers, defaults, and propositional attitudes).

As shown in Table 4.9, only the work of Abzianidze(2015; 2016; 2017b) adapts, in addition to natural logic, a proof search mechanism of a formal language (analytical tableau) to chain over multi-premises. Hu et al.'s 2019b system does handle multi-premises as well but only by relying upon replacement procedures and a number of ad hoc inference rules, as the examples in Table 4.7 show. Their work has only covered the first section of the FraCaS data sets (generalized quantifiers). Thus, it is not clear how far and how well replacement procedures could perform with respect to multi-premises of other phenomena. It is also likely that forward-chaining algorithms of the kind described will become very slow when the number of replacement rules expands. Judging by their performance on generalized quantifiers only, the natural tableau of Abzianidze is still ahead of Hu et al.'s order-based approach with 93% accuracy vs. 88%. Natural tableau of Abzianidze(2015; 2016; 2017b) and NatLog of [MacCartney, 2009] handled more semantic phenomena from the FraCaS dataset than generalized quantifiers, including propositional attitudes. However, we believe that the treatments of all these systems have some limitations, in particular:

- Among quantifiers, we take '*most*' in, for instance, '*most birds can fly*' to impose a default proposition: if X is a bird then X can fly **unless** there is something

that says it cannot. The same goes for the quantifier '*few*' in '*few birds can fly*', meaning that the default is that birds cannot fly unless there is an exception. However, the systems reviewed in this chapter have all treated '*most*' and '*few*' as generalized quantifier which interpret the above sentences roughly as *more than half of birds can/cannot fly.* Such interpretation is not wrong, but it cannot support the kind of reasoning (non-monotonic) described above (see Section 5.2.2 for further discussion of defaults).

- All the systems described above have deliberately kept the tense and aspect information out of consideration. For instance, NatLog assigns the **equivalent** ($\equiv$) entailment relation (when aligning) between, e.g. '*did sleep*' $\equiv$ '*has slept*' and '*is sleeping*' $\equiv$ '*sleeps*'. MonaLog [Hu et al., 2019b] adds rewrite rules such as *have* = *has* as part of knowledge-base, meaning replacing one by the other will generate an inference, and so on. Although their choices were motivated by the RTE task guideline, which explicitly states that tense and aspect information are to be ignored, such a choice comes at a cost: categorising invalid inferences as valid starting from simple arguments such as '*John loved Mary.*' $\models$ '*John loves Mary.*' to arguments of propositional attitudes where tense agreement between the attitude verb and its complement is a key such as '*John managed to eat a peach*' $\models$ '*John will eat a peach.*' which is clearly wrong (more discussion of this is in Section 5.8.2). NatLog and MonaLog have also ignored the distinction between singular and plural, leading to the derivation of '*Mary loves some men*' from '*Mary loves some man*'.

- None of these systems consider entailments involving propositional attitudes in any depth. Although some of them have ad hoc rules for dealing with the few cases of propositional attitudes that occur in the FraCaS dataset, none of them provide a systematic approach to this set of phenomena. There is a substantial literature dealing with the semantics of propositional attitudes, but very little of this has filtered through to implemented NLI systems. We believe that these are important issues, and we have made a considerable effort to deal with them (Section 8.5).

Natural Logic base Inference system	Premises		Derivation grammar	MR	polarity marking	BK	Inference rules	Proof Method-/Engine	Evaluation on dataset
	single	multi							
MacCartney and Manning [2007, 2008, 2009]	Y	N	Stanford statistical parser	Phrase-structure trees	Bottom-up & Tregex patterns	WordNet lexical relations		NatLog, Proof by alignment	On sections (1, 2, 5,6 and 9) of the FraCas with an average 87% accuracy and 59 % on the RTE3 problems.
Hu et al. [2019a]	Y	Y	C&C and Easy CCG	Polarized NL texts	Polarizing tool Hu and Moss [2018]	A set (K) of order-relations ($x \leq y$) s.t. x and y of any type: imported from WordNet, hand-coded orders about quantifiers and of specific linguistic construction in p_s or h	Generate and search: generate and update the set S of inferences and contradiction from p_s, then (depth-first) search a text-to-text match of h	rules of replacement and some others	On the first section of the FraCas with an average accuracy 88 %
Hu et al. [2019b]	Y	Y	C&C and Easy CCG	Polarized NL texts	Polarizing tool Hu and Moss [2018]	A set (K) of order-relations ($x \leq y$) s.t. x and y of any type: imported from WordNet, hand-coded orders about quantifiers and of specific linguistic construction in p_s or h	Generate and search: generate and update the set S of inferences and contradiction from p_s, then (depth-first) search a text-to-text match of h	rules of replacement and some others	The test part of the SICK dataset with 77.19 % average accuracy
Muskens [2010]	Y	Y	N/A	LLF	No explicit marking algorithm; via signed tableau entries and monotonicity rules	assumed hyponym relations between words	Around 20 specifically tailored rules	Natural Tableau	N/A (theory only)

Abzianidze [2014, 2015]	Y	N	C&C and Easy CCG	LLF	No explicit marking algorithm; via signed tableau entries and monotonicity rules	-WordNet lexical relations(synonyms, hyponyms and hypernyms no WSD) and Signature (words with algebraic features e.g. monotonicity)	Inventory of 50 rules of which: 20 Muskens's and the rest manually collected from the trail part of SICK dataset	automatic theorem prover: LangPro	test part of SICK dataset with accuracy 82.1 %
Abzianidze [2016, 2017a]	Y	Y	C&C and Easy CCG	LLF	No explicit marking algorithm; via signed tableau entries and monotonicity rules	-WordNet lexical relations(synonyms, hyponyms and hypernyms no WSD) and Signature (words with algebraic features e.g. monotonicity)	Inventory of 80 rules: the 50 rules of Abzianidze [2014, 2015] and the rest are manually collected from the FraCas dataset	automatic theorem prover: LangPro	On section 1, 2, 5 and 9 of the FraCas dataset with 87.1 % accuracy

Table 4.9: Classification of natural logic based inference systems.

Chapter 5

Issues in Non-Lexical Semantics

Following the natural logic tradition, a question we aim to answer in this thesis is (restated from Section 1.3): Q1 *"Can we use representations that are close to trees obtained by standard syntactic analysis as the MRs for complex issues in semantics?"*

Before attempting to answer this question, we first discuss several phenomena that are believed to contribute to complex semantics, particularly those which are often a source of debate and pose some challenges in the traditional (FOL) theorem-proving systems (e.g. quantifiers, defaults, time and aspect and propositional attitudes) [Bos, 2008]. For each phenomenon, the discussion is mainly two-fold. First, we discuss the way the phenomenon has been dealt with in the literature. Second, we present the aspects we have considered from the latter to capture the phenomenon in our MRs, keeping in mind the need to: 1) carry out inference over examples involving the phenomena in question, and 2) bridge the literature gaps (discussed in Section 4.4) in regards to defaults, time information, the distinction between singular and plural, and propositional attitudes. These semantic considerations are used then to feed the procedural steps of our inference system's main parts (CO2): the dependency parser (Chapter 6), the trees normalizer (Chapter 7), and the theorem prover (Chapter 8).

5.1 Notational Conventions

Before engaging in the discussion of semantic phenomena, this section explains some used notations in some examples. Mainly, most of the examples are instances of the following:

A general structure

(Operator , X :: {...,X})

Operator: binding operator; X: binding variable; ::: binding symbol; {...,X}: restrictor

However, as we go, the restrictor gets complicated as what gets bound here is a tree part, not a predicate (as in formal logic). So here is some instances of the `restrictor` and the above general structure with their meaning:

(1)	`[den(man)]`	the set denoted by `man`
(2)	{`name[den(John)]`}	the set denoted by the name `John`
(3)	`(forall,X::{[den(man)],X})`	the set of all `man`
(4)	`(forall,X::{[den(man)]`**`@@T`**`,X})`	the set of all `man` **at the time given by `T`**
(5)	`(exists,X::{[den(man)]@@T,X}`**`&{card,X,=3}`**`)`	the set denoted by `man` at the time given by `T` **and its cardinality `(card)` equals to 3 `(=3)`**

(6)	`(exists,X::{[den(man),`
	`modifier(amod(simple),`
	`*(+),[den(Italian)])``]`
	`@@T,X}&{card,X,=3})`
		the set of three `man` who are `Italian` at the time given by `T`

(7)	`..., exists, E, ({[den(walk),`
	`arg(subject, ..., [den(John)])]@@`
	`{simple,T},E})`
		A `walk` event where the the subject of walking is `John` at `simple` time given by `T`

Each part of the above examples is to be explained in later chapters. Among which, the term `modifier/3` in (6), which as shown in Figure 6.14(b) and illustrated in Section 6.3, consists of 3 parts: the modifier `([den(Italian)])`, the grammatical label `(amod(simple))` (i.e. a simple adjective —see Table 6.3 for the list of modifiers possible grammatical labels), and a sign `(*(+))` as an indication of its semantic class (see Section 5.4 for its discussion) which is in the above example *intersective*. As can be noticed above, all of the examples are of nouns. Verbs, motivated by choice of Davidson's (1967) treatments for events (see Section 5.5), have a slightly different shape which is:

Verbs

`(..., exists , E , {...,E})`

`exists`: binding operator; `E`: event variable; `{...,E}`: restrictor

Restrictor: {`[Verb, Args]@@Context,E`}

Verbs can have number of arguments and/or modifiers (e.g. '*John walks slowly*'). Each argument a verb has is expressed using the term `arg/4`, see Figure 6.14(b) and Section 6.3, which consists of parts as well one of which is the grammatical label (e.g. `subject` in example (7) of the above table). As mentioned above, these examples are

to be explained at length in later chapters, and the given notational illustrations we believe would suffice for the readability of the remaining of this chapter.

5.2 Generalized Quantifiers in Natural Language

Motivated by FOL's lack of expressiveness for complex mathematical quantifiers, the *Generalized Quantifiers* (GQs) theory has been introduced[1] by Mostowski [1957] and Lindström [1966] in mathematical logic as an attempt to go beyond FOL $\forall$ and $\exists$ operators. The theory found its way to the semantics of NL first, implicitly without direct mention of GQs, in Montague's grammar [1974] in which he interpreted NPs as type $\langle 1 \rangle$ GQ (see below). However, it is Barwise and Cooper [1981] who introduced the very first, and most influential, article that observed and hence illustrated how GQs naturally interpret NPs and quantificational determiners [Keenan and Westerståhl, 1997; Westerståhl, 2013].

In the theory of GQs, determiners constituting quantifier expressions, such as '*some*', '*at least four*', '*every*' and '*three...*', are mainly expressed as type $\langle 1,1 \rangle$ quantifiers. In a type $\langle 1,1 \rangle$ quantifier the 1 means a 1-ary relation i.e. a denoting set of individuals [Westerståhl, 2013]. That makes a type $\langle 1,1 \rangle$ quantifier Q in $(Q\ A\ B)$ representing a binary relation between two denoting sets A and B, in which Q is only sensitive to the cardinalities of the *restrictive set* (A) and the *intersective set* ($A \cap B$) [Alshawi and van Eijck, 1989]. In model theory, denoting sets range over a domain (let us say D). However, in a linguistic context, D can be thought of as the domain of discourse that lies in the background of an utterance [Westerståhl, 2013]. For example, the utterance '*some professors are smart*'. It might convey a set of people at a particular university as the domain of discourse (D) such that: the denotation of '*professors*' (the set A) is the subset of D that includes the professors at that university [Westersthl, 2019]; Those who are '*smart*' is the subset B of anyone smart at the same university, and the quantifier '*some*' in that utterance express the fact that the intersection between the set of '*professors*' and those holding the property of being '*smart*' is a *non-empty* set (i.e. of cardinality ≥ 1). The interpretation of quantifiers as relations between sets, a.k.a *second-order relations*, allowed studying some of their mathematical properties that includes: *symmetry*, *transitivity* and *conservativity* as well as *monotonicity* [Van Benthem, 1984]. Since monotonicity has been discussed at length in Chapter 4, Table 5.1

[1] There exist many surveys of the history of GQs that go back to the quantifiers of Aristotelian syllogisms, e.g. Westerståhl [1989]; Keenan and Westerståhl [1997] and an article in *The Stanford Encyclopaedia of Philosophy* written by Westersthl [2019].

illustrates the inference schemes that a quantifier Q ought to satisfy for each of the other properties.

Property	Inference Scheme	Example
Symmetry	$\dfrac{Q\,A\,B}{Q\,B\,A}$	$\dfrac{\textit{Some cats are animals}}{\textit{Some animals are cats}}$
Transitivity	$\dfrac{Q\,A\,B \quad Q\,B\,C}{Q\,A\,C}$	$\dfrac{\textit{All cats are animals} \quad \textit{All animals are mortals}}{\textit{All cats are mortals}}$
Conservativity	$\dfrac{Q\,A\,B}{Q\,A\,(A \wedge B)}$	$\dfrac{\textit{Every cat is an animal}}{\textit{Every cat is a cat who is an animal}}$

Table 5.1: Some mathematical properties of type $\langle 1,1 \rangle$ GQs.

In this thesis, we follow common practice in representing the meaning of *noun phrase*s (NPs), such as '*three cars*', '*every man*', '*John*', '*cats*', etc., of which quantificational determiners might be a part, as in the first two examples. In the theory of GQs, NPs "are most naturally interpreted as type $\langle 1 \rangle$ quantifiers" [Westerståhl, 2013] as it is considered the most uniform and elegant way to the semantics of NL *subject-predicate* constructions [Van Benthem, 1984]. Therefore, the following sections illustrate from a set theoretic perspective what is it like to be a type $\langle 1 \rangle$ quantifier. The discussion is split in two parts: one for NPs that include quantificational determiners, and the other for those that do not, such as proper names and bare plurals. We conclude this section by showing how we intend to render quantified NPs in our MRs based on this discussion.

5.2.1 Quantified NPs

Quantified NPs (QNPs) are NPs that include a determiner such as '*some professors*' and '*three red cars*'. In GQ theory, they are represented, in any domain of discourse D, as a set of subsets of D, i.e. type $\langle 1 \rangle$ quantifiers. For example, the extension of '*three red cars*' is the set of subsets of D whose intersection with the set of '*red cars*' in D is of cardinality equal to 3 [Westerståhl, 2013]. Symbolically, we can say that a quantified phrase $Q\,A$ denotes the set $\{X \subseteq D \mid [\![A]\!] \cap X \text{ (obeys a certain condition)}\}$, in which $[\![A]\!]$ is the set of individuals in D denoted by A [Van Benthem, 1984]. Below are some examples of the quantified NPs and what they denote in every D [Van Benthem, 1984]:

(5.1) a. **All A**: $\{X \subseteq D \mid [\![A]\!] \subseteq X\}$.

b. **Some A**: $\{X \subseteq D \mid [\![A]\!] \cap X \neq \phi\}$.

c. **A/An A**: $\{X \subseteq D \mid \mid [\![A]\!] \cap X \mid = 1\}$.

d. **No A**: $\{X \subseteq D \mid [\![A]\!] \cap X = \phi\}$.

e. **Three A**: $\{X \subseteq D \mid \mid [\![A]\!] \cap X \mid = 3\}$.

f. **At most two A**: $\{X \subseteq D \mid \mid [\![A]\!] \cap X \mid \leq 2\}$.

Within this framework, *proper names* also have to be interpreted as GQs, rather than simply treating them as denoting individuals. Montague was the first to devise the machinery for such treatment, where a proper name such as '*John*' is denoted in every D as a set of singleton subsets of D of which the individual representing '*John*' is a member (*Montagovian individuals* [Westerståhl, 2013]):

(5.2) **John**: $\{X \subseteq D \mid [\![John]\!] \in X\}$.

Many languages allow NPs to be constructed with zero/implicit determiners. The interpretation of such NPs varies from language to language – in Arabic NPs without determiners are interpreted as though they had an implicit indefinite article, in Persion they are interpreted as having an implicit definite article. English *bare NPs* (BNPs), where the head noun is *plural* or *mass noun*, are particularly hard to treat because they seem to have different interpretations in different syntactic contexts [Cohen and Erteschik-Shir, 2002; Abbott, 2006]. Westerståhl [2013] suggests that NPs of that kind "can be roughly treated as if they had a null universal or existential quantifier". This suggestion is not very helpful, given that universal and existential quantifiers are entirely different. Consider the examples in (5.3), where the *bare plural* (BP) '*cats*' conveys a different quantifier in each sentence: (5.3a) seems to require having all '*cats*' for it to be true and hence conveys a universal reading, in (5.3 b) and (5.3 d) the existential reading some '*cats*' would suffice, while in (5.3c), '*cats*' has a generic reading that is a bit more relaxed than a universal. Such variation in readings can also be found in BNPs with *mass nouns* (MNs), notice the difference in reading that '*water*' has in the examples (5.3(e)) and (5.3(f)) Abbott [2006]. While it is clear what quantifier the existential and universal readings imply, it is very difficult to give a precise characterisation of what kind of quantifier is involved in a generic reading. A number of proposals for dealing with generic readings have been suggested in the literature, however they tend to run into counter-examples.

(5.3) a. Cats are animals.
b. Cats are everywhere.
c. Mary loves cats.
d. Mary owns cats.
e. Water with fluoride in it is good for the teeth.
f. There was water with fluoride in it in the test tube.

Handling BNPs and implementing a comprehensive theory that tackles their interpretation in all contexts is beyond the scope of this thesis. Instead, we focused on studying their properties within the scope of the FraCas examples we intended to tackle and then introduced solutions (see Section 7.3.1) which support the required inferences. These solutions were based on a general assumption that BNPs in subject positions have a universal reading, while as complements they are treated as existentials. That assumption was motivated by two observations. First, in the intended parts of FraCas, BNPs do not appear as complements of *habitual* verb phrases[2]. In habitual settings, BNPs tend to have generic readings [Ramsay, 1992]. For instance, '*peaches*' in (5.4b) is clearly generic; upon hearing (5.4b), one would feel confident that if John were offered a peach, he would eat it. In contrast, hearing (5.4a) implies that there are some other peaches that John does not eat. The difference between these two is fairly subtle. On the other hand, in the non-habitual examples (5.5) '*some peaches*' and '*peaches*' are similar in meaning and convey existential quantification; if John was eating peaches then, he was eating some peaches, and if he was eating some peaches, then he was eating peaches, so one would want to be able to prove (5.5a) from (5.5b) and vice versa. The second observation is that many FraCas examples are *existential sentences* (see Section 5.7) such as '*there are men...*'. Subjects of an existential sentence are always interpreted existentially [Abbott, 2006], thus, they are an exception to the assumption that subject BNPs are universally interpreted [3].

(5.4) a. John eats some peaches.
b. John eats peaches.

(5.5) a. John was eating some peaches.
b. John was eating peaches.

[2] A verb phrase that presents a regularly or repeatedly occurring action.

[3] We assume, as does Abbott, that the subject of an existential sentence is the NP **after** the verb, not the expletive '*there*'.

To sum up, while our general approach is proof theoretic rather than model theoretic, we do follow a number of the insights from GQ theory. In particular we believe that in order to infer, for example, '*a man loves at least one car*' from '*some man loves three cars*' NPs ought to denote sets. Thus, from GQ theory, we have adopted the following conventions:

- Since GQs are generalizations of the existential and universal operators of FOL, our MRs will have QNPs with existential interpretations such as '*a man*', '*some men*' and '*three men*', defined as instances of a single variable binding operator `exists` that has the cardinality condition as part of its restrictor: e.g. '*three men*': `(exists, X :: ({[den(man)]@@T,X} & {card,X,=3}))`.

- Natural logic inference systems such as NatLog and MonaLog take '*all men*' $\sqsubseteq$ '*some men*' and hence (in a positive polarity context) '*all men love Mary*' $\models$ '*some men love Mary.*'. In other words, these systems as well as the aforementioned set-theoretic interpretation of '*all A*' assume *existential import*[4] of *A*. In contrast to that assumption, we believe that, as in standard FOL, universal expressions '*all A*' might not contain any *A* in the domain of discourse, and thus they are better interpreted as rules whose conclusion can be inferred only if the set of premises contains at least one sentence that asserts the existence of some individual(s) that satisfy A. For example, given '*all men love Mary*' and '*John is a man*' it is possible to infer '*there is a man who loves Mary*' because the premise '*John is a man*' asserted the existence of a man named John. Consequently, our universal rules resemble, to some extent, the rules of FOL's $\forall$ operator, with '*all men...*' looking roughly like `(forall, B::{[man],B}⇒...)`.

- There are other quantifiers that we treat as rules as well, namely '*most*' and '*few*' as *default rules* (see Section 5.2.2), and '*no*' as a *negation rule* (see Section 5.5.2).

The detailed illustration of the steps of turning QNPs into GQs or rules is given in Chapter 7, but for now we will just note that choosing such a representation for NPs has made going from '*three cars*' to, for instance, '*two cars*', '*one car*', or '*at least two cars*' a matter of calculations on cardinalities which is more efficient than having them as axiomatised orders in the background knowledge (as done in the MonaLog system, Section 4.3.2).

[4]Existential import "is the principle that every term has a non-empty extension" [Van Eijck, 2005].

5.2.2 The treatment of '*most*' and '*few*' as defaults

In standard deductive reasoning (such in classical logic and FOL), adding new information to a set of premises p_s will not invalidate or change any of the previously drawn conclusions from p_s [Brewka et al., 1997]. For instance, given the true premises '*all men are mortals*' and '*Socrates is a man*', one would conclude, by simple syllogistic reasoning, that '*Socrates is a mortal*' and that conclusion will not be affected if more premises were added to the two above [Brewka et al., 1997]. A reasoning system with such property is called *monotonic*[5].

On the other hand, a sentence such as '*birds (normally) fly*' is weaker than '*all birds fly*' in a sense that the first carries a seemingly open-ended list of exceptions: '*penguins*', '*ducks*', etc [Bochman, 2007]. Put differently, what is inferred from such a sentence is considered a plausible conclusion based on what is **normally** the case. For example, learning that '*Tweety is a bird*', it is plausible to conclude that '*Tweety flies*'. However, adding the information that '*Tweety is a penguin*' and '*penguins do not fly*' states that the situation is *abnormal* in a sense and the previously drawn conclusion is refuted [Brewka, 2012]. This kind of reasoning is called *non-monotonic* and the sentences causing such behaviour are called *rules with exceptions* or *defaults* [Brewka et al., 2008].

Attempting to express rules with exceptions such as '*birds (normally) fly*' by a FOL formula as in (5.6b) by refining the formula of '*all birds fly*' (5.6a) to include a predicate for capturing all exceptions in a model, is not the way to go (as its practically one might find hard, if not impossible) [Brewka et al., 2008]. An alternative is defining a rule of what is normally the case, with some side conditions under which a conclusion might be retracted. In Reiter's [1980] *default logic*, which is considered one of the most important formalizations for non-monotonic reasoning[6] [Brewka, 2012], defaults are formalizations of such alternatives. Thus, Reiter's logic interprets the default sentence '*birds (normally) fly*' as the rule given in (5.6c) and reads: "if x is a bird and it is consistent to assume that x can fly, then we can infer that x can fly"[Reiter, 1980].

[5]This property of reasoning systems is not to be confused with the natural logic semantic property: monotone (↑)

[6] Important alternatives to default logic for non-monotonic reasoning includes *autoepistemic logic* [Moore, 1985], and *circumscription* [McCarthy, 1980] (all explained at length in Donini et al. [1990]; Brewka et al. [1997, 2008]).

(5.6) a. $\forall(x)(\text{BIRD}(x) \implies \text{FLY}(x))$

b. $\forall(x)((\text{BIRD}(x) \wedge \neg\text{ABNORMAL}(x)) \implies \text{FLY}(x))$

c. $\dfrac{\text{BIRD}(x) : \text{FLY}(x)}{\text{FLY}(X)}$

Reiter's default logic consists of a set of (propositional or FOL) formulas (W) representing what is **known** to be the case and a set (D) of defeasible rules – defaults – of what what is **normally** the case [Brewka et al., 2008]. Defaults are rules of the form shown in (5.7) [Sergot, 2007] in which: the *prerequisite* formula α represents a premise that must be true in order for the rule to be applicable [Brewka, 2012] and γ is the inferred conclusion. The consistency conditions $\beta_1, \beta_2, ..., \beta_n$ are the formulas that have to be consistent i.e. there is no information that $\neg\beta_1, \neg\beta_2, ..., \neg\beta_n$ hold.

(5.7) $$\frac{\alpha : \beta_1, \beta_2, ..., \beta_n}{\gamma}$$

It is believed that the sentence '*most birds fly*' shares the same semantics and inference behaviours as defaults, i.e. it carries possible exceptions. However, its interpretation as a type $\langle 1,1 \rangle$ GQ in (5.8), which basically states that more than half of birds fly, cannot support the kind of non-monotonic inference described above. We therefore give '*most birds fly*' the same reading as Reiter's defaults (see Section 7.5.5) rather than the GQ interpretation in (5.8).

(5.8) **Most A B**: $\{A \subseteq D \,\&\, B \subseteq D \mid\mid A \cap B \mid \geq \dfrac{\mid A \mid}{2}\}$.

Another type of QNPs, that we believe in a way fall under the defaults category and are better expressed as rules instead of GQs, are the ones with the quantifier '*few*' as a determiner. In English grammar, the difference between '*few*' and '*a few*' is pretty subtle. Consider the examples in (5.9). The QNP '*few students*' in (5.9a), indicates a very small quantity of students who passed the exam, and that in a way expresses the fact that '***most** student **did not** pass the exam*'. On the other hand, what '*a few students*' indicate is that there are students who did pass the exam (i.e. some of them), but still their number is not as many as those who did not.

(5.9) a. Few students passed the exam.

b. A few students passed the exam.

5.3 Definite NPs (DNPs)

5.3.1 Definite Descriptions

Definite and *indefinite* descriptions, are common terms that have been used in the philosophical literature [Abbott, 2006] to mark two prominent uses of NPs; '*the NP*' for the first and '*a/an NP*' for the latter. Uttering a sentence with an indefinite NP (e.g. '*A man*' in (5.10a) invites the addressee to note the existence of some individual that fits the description. Definite NPs, on the other hand, such as '*the man*' in (5.10b and 5.10c), instructs the addressee to **find** '*a man*' that fits the description. The thing to be found is often called the '*referent*' of the definite NP. Now the question remains is where it should be found.

(5.10) a. A man is sleeping.

b. (*Scenario: I walked into the room. There is a man lying on the couch*) The man is sleeping.

c. A man entered the room. The man is wearing blue jeans.

It is evident that the referent of '*the man*' (in the same examples above) is not the one and only man in the world at large, as Russell's [1905] *uniqueness theory* claims. Instead, and according to the *familiarity theory* [Christophersen, 1939; Heim, 1982, 1983; Roberts, 2003] it is the one and only one man that is known to both the speaker and addressee either by being previously mentioned in the conversation (as in (5.10c)) or by being contextually the most *salient* referent (as in (5.10b)).[Lewis, 1979; Heim, 1982; Von Heusinger, 2002]. These two uses[7] of definite NPs are often called *anaphoric* and *situationally salient* respectively [Von Heusinger, 2002], and both require the addressee's familiarity with the referent. There do exist other uses of the definite article '*the*', one of which is with NPs whose descriptive content is sufficient to point out an absolutely *unique* referent in the universe such as '*The moon*' in (5.11a). The *functional* use, as in (5.11b), is another. It picks up exactly one referent that is determined, based on other factors than the current conversation or the context in which it held such as time (i.e. president at what time?) [Von Heusinger, 2002]. In these two uses of definite descriptions, addressees are not required to be familiar with the referent. They would *accommodate* its existence instead. In other words, "addressees are willing to accept a definite description, if they are able to figure out the intended referent" [Abbott, 2006, p. 3].

[7]These uses of definite NPs and their associated terms are all based on Von Heusinger's [2002] survey of definite NPs.

(5.11) a. The moon is bright.

b. The president of the US is the most powerful person in the world.

Definite descriptions appearing in the intended sections of the FraCas test-set range over the four uses illustrated in Table 5.2. Therefore, the process of their resolution (see Section 7.7 and Figure 7.37) involves two routes:

1. Look for an item[8] which can be proved to satisfy the descriptor using information that is available in the minutes[9], and resolve the defining description to it; or

2. *Assimilate* its existence, assuming the addressee's familiarity with, or accommodation to, the assimilated referent. Then resolve the definite description to it.

As for their MR, indefinite descriptions, as seen in Section 5.2, have been turned into GQs, particularly as instances of the `exists` operator with their cardinality included. Similarly, definite descriptions have a representation that is similar to their equivalent indefinite version, but with the definite article as the operator. Notice, for example, the pattern of changes between '*three men*' and '*the three men*' in (5.12a and 5.12b), and '*some men*' and '*the men*' in (5.12c and 5.12d).

(5.12) a. **Three men**: `(exists, X :: ({[den(man)]@@T,X} & {card,X,=3}))`

b. **The three men**: `(the, X :: ({[den(man)]]@@T,X} & {card,X,=3}))`

c. **Some men**: `(exists, X :: ({[den(man)]@@T,X} & {card,X, ≥1}))`

d. **The men**: `(the, X :: ({[den(man)]@@T,X} & {card,X,≥1}))`

5.3.2 Proper Names

Proper names, such as '*John*' and '*Peter*' in (5.13) are considered definite NPs as well. It is plausible to say that, in most contexts, when they are used it is assumed that they have already been introduced to the addressee and have a unique designator [Abbott, 2006; Abbott and Geurts, 2002]. Therefore, for their resolution, we assimilate the first

[8]Note that, having multiple referents is possible and resolving a definite description in such case requires attempting to find the most salient one. Moreover, finding the most salient one is based on the *non-lexical context* within which the description is mentioned. In other words, it requires some *pragmatic* considerations that are beyond the scope of this work.

[9]We take the minutes to mean "the written record of what was said at a meeting" [Cambridge Online Dictionary, 2021]. The visible content of the minutes is what we assume the participant of a discourse share with no assumption of some privileged access to what the other person has inside their head.

Definite NP use	Example from the FraCas test suits
Situationally salient	**(fracas-039)**: Some delegates finished the survey.
Anaphoric	**(fracas-046)** Neither commissioner spends time at home. One of the commissioners spends a lot of time at home.
Functional	**(fracas-001)**: An Italian became the world's greatest tenor.
Unique	**(fracas-050)**: The North American continent ...

Table 5.2: The uses of definite NPs as surveyed by Von Heusinger [2002].

mention of the name and resolve any subsequent mentions in the same conversation to that one. For their MR, we have them turned into a form that supports their Montagovian treatment as GQ (given in (5.2)) and the fact that they are referring expressions and hence need to be resolved as following:

- A name N is turned into a representation that is equivalent to '*the named N*' [Geurts, 1997] (see Section 7.3.2). For example:
  John: `(the, X ::({name,[den(John)]}@@T,X} & {card,X,=1}))`
- As a GQ, a name N denotes a set of singleton subsets of which the individual denoting the name N is a member (i.e (5.2)). In other word, N denotes the of properties that the name N satisfies. For example: '*John is a man*', '*John is a person*', etc. This part we have hand-coded (whenever necessary)[10] in the background knowledge as a form of forward rules, such that assimilating the existence of the name into the minutes of the conversation will introduce its other properties as well (see Section 8.3.1 for detailed discussion).

(5.13) a. John loves Mary. He is a fool.
b. John loves Mary but Peter does not. He is a fool.

5.3.3 Pronouns

Lastly, when used in conversation, pronouns[11] also have an intended salient referent that the addressee is assumed to be familiar with. However, the description of the referent is not lexically apparent as in the case of definite description or proper names. For

[10] Although proper names appearing in FraCas did not need encoding such knowledge, for our running examples we needed to demonstrate how a pronoun such '*he*' in (5.13a) is resolved to '*John*'.
[11] Anaphoric personal pronouns in particular.

instance, the pronoun '*he*' is used to refer to to a singular male individual, while '*she*' refers to a singular female individual. Therefore, their resolution typically involves factors such as *number*, *gender* and *recency* [Tetreault, 2005]. The first two, we have incorporated in their MRs so that '*he*' and '*she*' are read as '*the singular male*' and '*the singular female*'. Recency, on the other hand, is used in the process of picking out the referent in case more than one are found. For instance, in (5.13b), recency will assign '*Peter*' as the referent for '*he*' and not '*John*'. Similar to definite descriptions, pronouns vary in uses. Other than the anaphoric use I have just described, there are the *deictic* and *bound* uses [Von Heusinger, 2002; Büring, 2011]. The first is similar the situationally salient use of '*the NP*', where a pronoun refers to the most contextually salient (that is not necessary the most recent) individual fitting the description, as in the example (5.14a). In the bound use, pronouns (such as '*his*' in (5.14b)) are acting as bound variables where the referent is a universally quantified NP. Pronouns, in general, are beyond the scope of the intended test suite, however, we have considered their anaphoric uses for demonstration examples where appear to be interacting with other definites ('*the NP*' and Names). Therefore, the other uses of pronouns will not be discussed any further (the reader can find some interesting discussions in e.g. Von Heusinger [2002]; Tetreault [2005]; Büring [2011]).

(5.14) a. (*Scenario: A man walked into the room*)
He is wearing a blue jeans.
b. Every man loves his car.

5.4 Adjectives

Adjectives form a word class whose main function is to modify nouns [Lalisse and Asudeh, 2015]. Syntactically, they can appear as part of the NP and be said to have an *attributive* role as in (5.15a), or as a copula complement (see Section 5.6) and hence have a *predicative* role as in (5.15b).

(5.15) a. John is an Italian man.
b. John is Italian.

What we would like to discuss in this section is their interpretation with respect to the nouns they modify and hence how that can be projected in their MR. A common semantic classification of adjectives is given in Figure 5.1 [Abdullah and Frost, 2005;

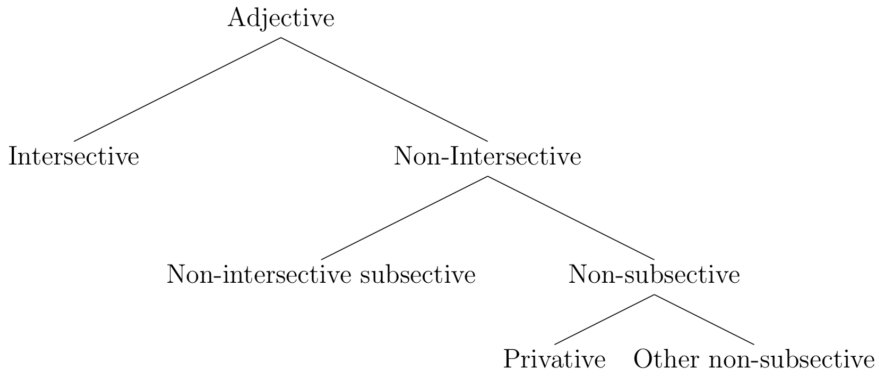


Figure 5.1: Types of adjectives [Lalisse and Asudeh, 2015].

Morzycki, 2013]. Among these classes of adjectives, we are particularly interested in the distinction between *intersective* and *non-intersective subsective* (subsective for short) adjectives. Intersectives have two main properties. First, when they combine with an NP, they create a modified NP whose denotation is the intersection between the set denoting the adjective and the set denoting the NP. For instance, the denotation of '*German professor*' in (5.16p_1), is computed as an interaction between the set of Germans and the set of professors (i.e. $[\![\text{German professor}]\!] = [\![\text{German}]\!] \cap [\![\text{professor}]\!]$)[12]. Second, from an inferential point of view, the adjective and the NP it modifies each can give rise to an independent entailment as shown in (5.16) [Morzycki, 2013]. The same cannot be said about subsective adjectives, because when they are applied to an NP, the resulting expression denotes a subset from the set denoting the NP [Lalisse and Asudeh, 2015]. For example, '*experienced professor*' in (5.17p_1) denotes a subset of the set of '*professors*' (i.e. $[\![\text{experienced professor}]\!] \subset [\![\text{professor}]\!]$). In addition, '*experienced*' is not truth-conditionally independent as is '*German*'. That would become apparent if we add '*John is a journalist*' as a valid premise to both (5.16) and (5.17), and then try combining each of the adjectives '*German*' and '*experienced*' with '*a journalist*'. In the first, we get a valid inference, since the fact of being a '*German*' can be added to the fact that John is a journalist. However, being '*experienced*' in (5.17p_1) is attributive to '*professor*' only, and we cannot use it independently in a sense that assumes that John is experienced at everything and hence is '*an experienced journalist*'.

[12]Note that '*German professor*' could also mean a professor who teaches German, like a mathematics professor, but this is not what we mean here.

(5.16) p_1. John is a German professor.
p_2. John is a journalist.
$\models$ John is a professor.
$\models$ John is German.
$\models$ John is a German journalist.

(5.17) p_1. John is an experienced professor.
p_2. John is a journalist.
$\models$ John is a professor.
$\not\models$ John is experienced.
$\not\models$ John is experienced journalist.

Now that we have established the distinction between intersective and subsective adjectives, the matter remaining is how they are going to be represented on the MR? First, based on their syntactic position:

- If they are part of the QNP they are modifying (i.e. attributive), then, for example: '*an **Italian** man*' would be:

```
(exists, X ::
 ({[den(man), modifier(amod(simple), *(+), [den(Italian)])]@@T,X}
 &{card,X,=1}))
```

- If they are copula complements, then are predicative. Thus, motivated by the intended treatment of copula sentences (see Section 5.6) they are represented as if they are attributive to some dummy invisible NP. For example: '*Italian*' in (5.16b) is represented as '... *Italian* _ ':

```
(exists, X :: {(modifier(amod(simple), *(+), [den(Italian)]@@T),X)})
```

Second, as can be noticed in the representation of adjectives above, there is some kind of sign included: $^*(+)$. This is the sign we use to distinguish between intersective adjectives $^*(+)$ and subsective adjectives $^*(-)$. These signs are assigned during the construction of the parse trees (see Section 6.2.2) and are eventually used to determine when an adjective can be detached from the nominal that it modifies and used as a separate additional fact in proofs, and when it cannot.

5.5 Events and Time

5.5.1 Events Semantics

In the standard logical view, a sentence with a transitive verb such as (5.18a), is represented, as it appears in (5.18b), as a relation (i.e. predicate) between the subject '*Jones*' and the direct object '*the toast*' [Maienborn, 2011]. Davidson [1967], pointed out that such representation does not allow an explicit reference to the action described by the sentence nor further characterize it by saying, for example: Jones did it '*with a knife*', '*in the bathrom*', '*at midnight*' [Davidson, 1967; Maienborn, 2011]. Thus, *Davidsonian event semantics* claimed that "action verbs introduce an additional hidden event argument that stands for the action proper" [Maienborn, 2011]. That turns the BUTTER relation in (5.18b) into a predicate of three arguments: the subject, direct object and the event argument. The latter is the variable e, which is bound by the existential closure $\exists\, e$ as shown in (5.19b). The introduction of the event argument e, has made expressing *adverbial modifiers* straightforward [Maienborn and Schfer, 2011]; i.e. as predicates, adding more specifications for the event, of which e is an argument. Davidson's classical example in (5.20a) is thus represented in the logical form in (5.20) in which adverbial modifiers are additional conjuncts [Champollion, 2014].

(5.18) a. Jones buttered the toast.
b. BUTTER(Jones, the toast)

(5.19) a. Jones buttered the toast.
b. $\exists\, e$[BUTTER(Jones, the toast,e)]

(5.20) a. Jones buttered the toast in the bathroom with the knife at midnight.
b. $\exists\, e$[BUTTER(Jones, the toast,e)$\wedge$ IN(e,bathroom) $\wedge$ INST(e,the knife)$\wedge$ AT(e, midnight)]

In the course of the evolution of Davidson's events semantics, two other directions have emerged showing some influential expansions to the original theory of event semantics and some other deviations [Maienborn, 2011]. The first direction is called *Neo-Davidsonian* event semantics, initiated by Higginbotham [1985, 2000] and Parsons [1990, 2000], which separates Davidson's verbal arguments such that the verb predicate only has the event argument, while the participants are linked to the event by the use of *thematic roles* [Maienborn, 2011]. As shown in Table 5.3 (c), the subject

'*Jones*' has the thematic role AGENT and the direct object '*the toast*' is the THEME. The Neo-Davidsonian paradigm also assumes a broader notion of events, that was given a term *eventuality* by Bach [1986], such that not only actions proper introduce event arguments, but any verbal predicate[13]. Moreover, the contemporary approaches to Neo-Davidsonian events semantics consider event arguments to be part of not only verbal predicates but also of predicates of any syntactical category (including nouns, adjectives and prepositions) [Maienborn, 2011].

The second direction is Kratzer's [1995; 1996; 2000] event semantics that argues that verbal predicates do introduce event arguments, nonetheless they only include, in addition to *e*, themes arguments (see Table 5.3(d) for example) [Champollion, 2014]. Furthermore, Kratzer [1995] has split predicates into parts that take event arguments and others that do not, based on the so-called *stage-level/individual-level* distinction[14].

a.	Standard FOL	BUTTER(Jones, the toast)
b.	Davidson	$\exists\, e$[BUTTER(Jones, the toast, e)]
c.	Neo-Davidson	$\exists\, e$[BUTTER(e)$\wedge$AGENT(e, Jones)$\wedge$ THEME(e, the toast)]
d.	Kratzer	$\exists\, e$[AGENT(e,Jones)$\wedge$BUTTER(e, the toast)]
e.	This work	`..., exists, E, ({[den(butter), arg(dobj,...), arg(subject, ...)]@@ {simple,T},E})`

Table 5.3: The representation of verbal arguments with respect to verbal predicates in different versions of event semantics (including this work).

Our representation of events is based on the original Davidson program, simply because, as shown in Figure 5.2, verbs in the parse tree are linked to their arguments via syntactical relations (i.e. subject, direct object, etc). And since our main goal for this thesis is to provide MRs that are as close as possible to the syntactic form whilst straightforwardly supporting inference, we are content with just adding the event argument as part of the verbs clause (as shown in Table 5.3e) to benefit from the ease of adding more characteristics (mainly time and adverbial expressions) to the event expressed by the verb.

[13]They are normally classified according to Vendler [1967] for situations into: *accomplishments*, *achievements*, *process* and *states*, see [Mourelatos, 1978; Filip, 2011] for examples and further discussion on their properties.

[14]Broadly speaking, stage-level predicates indicate temporary or accidental properties such as '*speak*', '*wait*', and '*tired*', while individual-level predicates, as '*love*', '*know*', '*intelligent*', and '*blond*', express permanent or inherent properties [Maienborn, 2011].

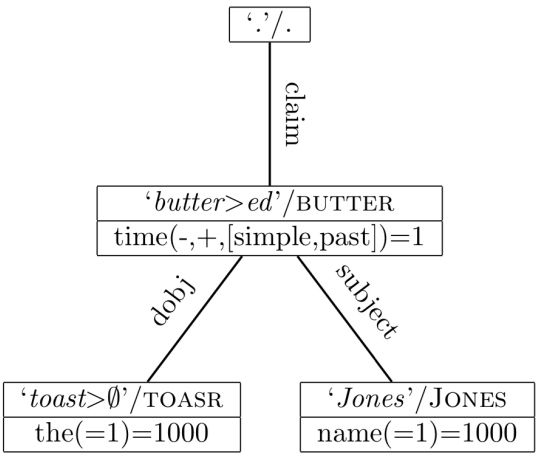


Figure 5.2: The dependency tree of '*Jones buttered the toast*'.

5.5.2 Events and Negation

As discussed in Chapter 3, our proofs are conducted constructively such that a proof of a negated proposition $\neg P$ is proceeded by assuming P and deriving that it leads to absurdity or falsehood $\bot$. In this thesis, we are looking at two sources of negation; the quantifier '*no*' (e.g. (5.21a)) and the negation operator '*not*' (e.g. (5.22a)). Choosing to introduce Davidsonian's existentially quantified event argument e in our MR invited some scoping considerations. That is, when interacting with other scope-bearing quantifiers, the chosen scope of $(\exists e)$ ought to reflect its intended reading. That scope is generally the narrowest possible, as argued by [Champollion, 2010; Herburger, 2011; Bernard, 2018] (among others). Notice the difference in reading when the event quantifiers had a narrower scope than the negation in ((5.21b) and (5.22b)) [Champollion, 2010], and when they had a wider scope over negation in ((5.21c) and (5.22c)) [Champollion, 2010]. In the latter, both examples' readings state the existence of an event other than sleeping that a man undergoes, which is evidently not true [Champollion, 2010; Herburger, 2011; Bernard, 2018]. More discussion on scope and their resolution is given later on this chapter (see Section 5.9).

(5.21) a. No man sleeps.

b. $\neg\exists x[\text{MAN}(x) \wedge \exists e[\text{SLEEP}(x,e)]]$

(Reads: there is no sleeping event that is done by a man.)

c. $\exists e[\neg\exists x[\text{MAN}(x) \wedge [\text{SLEEP}(x,e)]]$

**(Reads: there is an event which is not a man-sleeping event.)*

(5.22) a. Some man did not sleep.

b. $\exists x[\text{MAN}(x) \wedge \neg\exists e[\text{SLEEP}(x,e)]]]$

(Reads: there is a man who is not involved in a sleeping event.)

c. $\exists x[\text{MAN}(x) \wedge \exists e\neg[\text{SLEEP}(x,e)]]]$

**(Reads: there is a man and an event which is not him sleeping.)*

5.5.3 Tense and Aspect

The natural logic-based inference systems discussed in Section 4.3, all lack any representation of time and that is motivated by the fact that the challenge they are addressing (the RTE task initiated by [Dagan et al., 2005]) "explicitly specifies that tense to be ignored" [MacCartney and Manning, 2009]. Nonetheless, as a consequence, that led to some wrong inferences about quite simple problems. For instance, NatLog (Section 4.3.1) considers '*sleeping*' ≡ '*sleeps*' [MacCartney and Manning, 2009] and the natural tableau (Section 4.3.3) will allow entailing '*John loves Mary*' from '*John loved Mary*'.

On the other hand, Bos [2008] argues, in his discussion of the semantic ingredients that ought to be part of a deep MR, that the least a MR should encode is "the time at which (or during which) the descriptive content of the sentence in question holds" [Ogihara, 2007] i.e. *tense*[15]. Consider, for instance, the sentences in (5.23a) and (5.23b). They both convey an event of John working, but differ with respect to the tense *morpheme*s ('*-s*' and '*-ed*'). That difference places the event at different times, present in the first and past in the second, at which it is true. Such analysis can be extended to more complex tenses using *aspect* constructions [Bos, 2008]. Aspect constructions express how an event unfolds over time: i.e. whether it is complete, in progress, frequent, etc. [DeCapua, 2008]. English aspects are '*simple*', '*progressive*' and '*perfective*' and are, as exemplified in (5.23a), (5.24a) and (5.24b) respectively, indicated by either main verb morpheme or the *auxiliary verb*+ main verb morpheme.

(5.23) a. John works.

b. John worked.

(5.24) a. John is working.

b. John has worked.

[15]The discussion of time here assumes the minimal clause, i.e. we do not consider overt nor covert temporal adverbs such as '*yesterday*', '*once*', '*every Sunday*', etc. which is believed to contribute to (if not to be the main carrier of) the temporal information conveyed by a sentence [Ogihara, 2011].

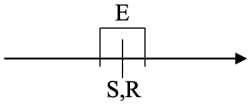
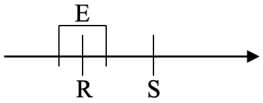
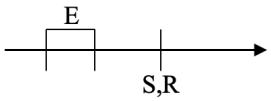
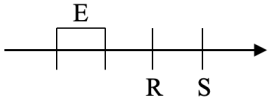
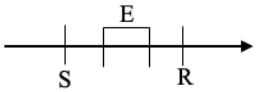
With that being said, following the choice of Davidsonian-style event representation, we further make advantage of such a choice and provide the temporal information conveyed by a sentence as part of that representation. Again, for the sake of staying relevant to the syntactical analysis of sentences, we rely on the auxiliary sequence plus the tense morpheme on the main verb to build up a route from the time when the sentence is uttered to the time when the event, expressed by the verb, took place (see Section 6.2.2). This particular machinery actually goes back to the system of Reichenbach [1947], which claims that a correct account for the meaning of NL tenses involves understanding the temporal relations that are held between three points in time: the *speech time*, the *event time*, and the *reference time* [Blackburn, 1994]. In a conversation, while it is clear that speech time refers to the time when the tensed sentence is uttered and event time is when the event actually took place, the reference time is not overt and is taken to be mutually known by the participants in the conversation for it to make sense [Van Lambalgen and Hamm, 2008; Hackmack, 2015]. Van Lambalgen and Hamm demonstrated further the concept of reference time through an example of a present perfect sentence '*I have caught the flu*'. Stating that, although the event of catching a cold is located in the past and the sentence is uttered in the present, it is also meant for the sentence to have some present relevance (say for example the speaker is constantly coughing) that both the speaker and the person addressed acknowledged. Therefore, the reference time of '*I have caught the flu*' is simultaneous with the speech time. With a similar mindset, Reichenbach argued that all other possible tenses are definable, in terms of the three time points and the *simultaneity* and *precedence* relations [Declerck, 1986; Hackmack, 2015]. Thus, given that the letters R, E and S stand for reference time, event time and speech time respectively, and that a simultaneity relation is represented by a comma (,) and that precedence relation as a dash interval (—) on the time axis, (5.25) summarizes Reichenbach's representation of the various tense expressions. For instance, for the simple past sentence in (5.25b), both E and R occur (simultaneously) before the speech time S.

(5.25)

a. John works.	*simple present*	S,R,E
b. John worked.	*simple past*	E,R —S
c. John will work.	*simple future*	S—E,R
d. John has worked.	*present perfect*	E —R,S
e. John had worked.	*past perfect*	E —R —S
f. John will have worked.	*future perfect*	S—E —R

It can be noticed that the above examples do not include progressive tenses. Reichenbach argues that a progressive tense, or *extended tense* as he called it, is taken to "indicate that the event covers a certain stretch of time" [Hackmack, 2015]. Therefore, progressive tenses are represented on the time axis as following [Hackmack, 2015]:

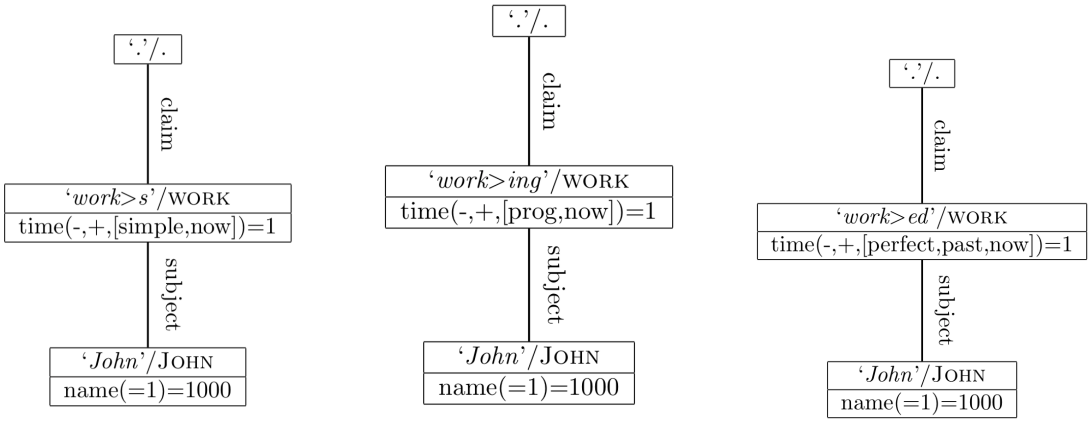
(5.26)

a. John is working.	*present progressive*	
b. John was working.	*past progressive*	
c. John has been working.	*present perfect progressive*	
d. John had been working.	*past perfect progressive*	
e. John will have been working	*future perfect progressive*	

Against Reichenbach's claim, [Comrie et al., 1985], who refined the Reichenbach system, and [Gamut, 1991] argue that Reichenbach's system does not offer a complete account for all temporal constructions demonstrating that sentences such as '*Mary would have sung*' cannot be expressed with only one point of reference [Gamut, 1991]. Moreover, Comrie et al. [1985] stated that reference time is irrelevant for simple tenses as they can be sufficiently expressed using the time of speech and event. The first argument concerns constructions that are not in the scope of this thesis (the reader can refer to Comrie et al. [1985] or Gamut [1991] for further discussion). The latter, is something we agree with and we further extend to the following conventions:

- We take that each inference problem (premises and a question) constitutes an ongoing (i.e. current) conversation. Therefore, the speech time for all of that problem's sentences is set to a default non-overt *now* tense.
- Agreeing with Comrie et al. [1985], speech time and event time are sufficient to express simple tenses. However, since speech time is the default non-overt present tense, only event time will be part of the temporal information.
- Other tenses will have the relation between reference time and event time as an ordered list that we refer to as the *tense sequence*.

- To distinguish between the simple and perfect tenses (5.25), and the progressive ones (5.26), a label (*simple*, *perfect* and *prog*) for aspect is added to the tense sequence.
- In Prolog notation, the tense and aspect information, as the examples in Figure 5.3[16] show, are all part of one ordered list, in which the aspect is the head and the tense sequence (event time **then** reference time (if any)) is the tail: `[Aspect|Tense_sequence]`.



(a) John works. (b) John is working. (c) John has worked

Figure 5.3: Examples of the temporal information as they appear in our pre-processed dependency trees.

5.5.4 Definiteness of Tenses

As the above examples show, the *time specifier* on the main verb maintains, in addition to the tense and aspect, some other features. One of these is what we call the definiteness sign. We use that sign to distinguish between specific points in time (+) and some unspecific points in time (-), akin to the difference between definite and indefinite NPs. Most scholars agree that the present tense denotes **the** point of speech [Von Stechow, 2009], hence, its definiteness sign is commonly (+). However, the semantics of past tenses has been a centre of debate. Among the existing formalizations of the semantics' effects on tense [Ogihara, 2011], there is an approach that assumes an existential quantifier interpretation for tense operators. This approach is attributed

[16]These trees are after the pre-processing step, explained in details in Section 7.2.1, that includes analysing the trees of tensed sentences into tenseless ones (where auxiliaries and tense morphology are removed) and a time specifier on the main verb.

to Prior's [1957; 1967] *tense logic*, in which the tense of an expression α is expressed by means of operators: **P**α "it has been the case α" and **F**α "it will be the case that α" [Ogihara, 2011]. To demonstrate, consider the example of Ogihara [2007] given in (5.27). The sentence (5.27a) is represented as a tense operator and a tenseless expression (5.27b) and is interpreted (5.27c) as an event of Kim leaving at an indefinite time in the past.

(5.27) a. Kim has left.

b. P [leaves(Kim)]

c. There is a past time at which Kim leaves.

On the other hand, referential approaches to tense, which were initiated by Partee [1973] and Enç [1987], argue to the contrary. The claim is that a past tense morpheme refers to a contextually particular salient time in the past and not to some non-specific time in the past, and that tenses in general are to be considered as referring expressions that are on par with pronouns [Ogihara, 2007, 2011]. Some other approaches, such as that of Ogihara [1996], argue for a mixed approach; an existential quantification over past times with a contextual restriction on the quantification force (e.g. $\exists t(t \subseteq \text{last year} \wedge Event(arg1, arg2, t))$)[Ogihara, 2011, 2007]. In the scope of this work we, assume a referential approach for both present and past tenses. Given all of the above, our version of the buttering event in Table 5.3 (e) will become, after including the temporal information, as follow (see Section 7.3 for further discussion):

```
the, T::{{time,[past]},T},
   exists, E, ({[den(butter), arg(dobj,...), arg(subject,...)]@@{simple,T},E})
```

5.6 Copula verbs

Sentences of forms like those in (5.28) are called *copula sentences* and they are, as Mikkelsen [2011, p. 1805] defined them, "a minor sentence type in which the contentful predicate is not a verb, but some other category like AP[17], NP or PP[18]".

[17] Stands for adjective phrase.

[18] Stands for prepositional phrase.

(5.28) a. John was a happy man.

b. The man who loves Mary is John.

c. The morning star is the evening star.

d. The man is in the park.

e. John is ugly.

Take for example, the non-copula sentence '*John loved Mary*'. The verb '*loved*' is the *predicator* of the subject '*John*'; '*Mary*' is what undergoes the event of being '*loved*'. Thus, in a FOL-like notation, the predicator of that sentence will be translated to something like LOVE(John, Mary). On the other hand, the predicator in a copula sentence such as '*John was happy*' is not the verb '*was*', it is the post-copula phrase, '*happy*'; HAPPY(John). Therefore the general semantic contribution of copulas (short form just copulas) are believed to be confined in either bearing time information of the predication or specifying an equational or identity relation between the subject and the post-copula [Heycock and Kroch, 1998]. For instance, in (5.28a) '*was*', states that there is a past time in which '*John*' satisfies the property of being '*a happy man*'. While the copula '*is*' in (5.28b) specifies that '*John*' is the identity of '*the man who loves Mary*' and in (5.28c) is stating that '*the morning star*' and '*the evening star*' are the same thing. However, simply treating such sentences as identities ignores the fact that equational or identity relations are also time bound. For instance, consider the copula sentence '*Mary's husband was John.*', in which the identity of '*Mary's husband*' was '*John*' at that past time, and we can not infer that he still is as they might now be divorced!

English copulas, in their common use, are in the form of the verb '*be*', ('*am*', '*is*', '*are*','*was*' and '*were*') as in the examples below. There are, however, other verbs that function as copulas, such as the verb '*become*' in '*John became a great chef.*'

One goal of this thesis is to do inferences. In proof theory, the equality of two items means that both are actually co-referring to the same object and that allows what is called *substitutability* in proofs [Fitting, 1990, Ch.8] i.e. any sentence that is true for one item, must be true for the other. Thus, despite the various ways copulas been classified in the literature, we believe, from the inferential point of view, that copulas of the form '*NP be NP*' are clearly representing equality relation between the subject and post copula at the time given by the copula verb. To demonstrate, consider the inference example in (5.29), given that '*John is a fool*' and '*every fool loves Mary*' it is a provable consequent that '*John loves Mary*' since '*John*' and the individual referred

to by '*a fool*' are the same thing. Now, consider the examples in (5.30), where p_1 indicates an equality relation between two NPs (the referent of '*John*' and '*a man*'), and p_2 predicates a locative feature '*in the park*' of the subject '*the man*' (whose referent is the man mentioned in p_1). Intuitively or by transitivity [Fitting, 1990, Ch.8], given that '*John*' is '*a man*' and the latter is the referent of '*the man*', one would accept that '*John is in the park*' as a valid consequent. Moreover, semantically, we believe that, for copulas of the form '*NP be PP*' such as the later p_2 and '*NP be AP*' such as '*John is Italian*', saying that the post copula is a predicate that the subject satisfies at the time given by the copula, is pretty much the same as saying that the individual designated by the subject and the individual satisfying the predicate at the time given are the same. Therefore, a step in constructing the MR of copula sentences involves transforming the syntactical analysis of '*NP be PP*' and '*NP be AP*' copula sentences into instances of '*NP be NP*', by saying both PP and AP are modifying an indefinite hidden NP and hence expressed roughly as '*NP be _ PP.*' and '*NP be _ AP _.*' (see Section 7.2.2).

This proof-theoretical concept of treating copula sentences as equalities was also adopted by the natural tableau (Section 4.3.3) in which their "rules for the copula resemble the FOL tableau rules for equality" [Abzianidze, 2017b].

(5.29) p_1. John is a fool.
p_2. Every fool loves Mary.
$\models$ John loves Mary.

(5.30) p_1. John is a man.
p_2. The man is in the park.
$\models$ John **is** in the park.

5.7 Existential Sentences

An *existential sentence* is "a specialized or non-canonical construction which expresses a proposition about the existence or the presence of someone or something." [McNally, 2011]. Put differently, sentences such as in (5.31) [Moro, 2017] are all to explicitly or implicitly state the existence of some entities. However, the term existential sentences (ES) is not intended for the canonical subject-predicate sentences such in (5.31a). It is used to refer to the specialized syntactical structures that generally have the form

'*There be NP Coda Phrase*' [McNally, 2011] as in (5.32) and share some specific characteristics[19]: (1) the expletive[20] subject '*there*'; (2) a verb '*to be*'; (3) a *pivot* NP that denotes the individuals whose existence or presence are under discussion [McNally, 2011]; and most commonly (4) a *coda phrase* that is typically an PP (e.g. (5.32a)) or an AP (e.g. (5.32b) and (5.32c)).

(5.31) a. Unicorns exist.
b. There are unicorns.

(5.32) a. There are men in the park.
b. There is a man sleeping.
c. There is a man who is sleeping.

How theses parts of ES relate syntactically and semantically, has been a centre of debate among linguists. The standard theory (which goes back to the Milsark [1974, 1977] and Stowell [1978]) is that '*there*' is a subject place holder that has no semantic content i.e. expletive [Moro, 2017; Abzianidze, 2017b] and the predication of the copula '*be*' is the coda phrase in which the pivot NP is an argument. This means that the ES in (5.33a) and the copula sentence in (5.33b) share the same underlying structure and their semantic representation would be as in (5.33c).

(5.33) a. There are men in the park.
b. Men are in the park.
c. IN_THE_PARK(men).

However, Williams [1984] and Hazout [2004] among others, argued that, although still semantically vacuous '*there*' is not a place holder for the misplaced subject; '*there*' is the subject [Moro, 2017]. Therefore, the pivot NP is regarded as the predicate and the coda phrase serves as its internal modifier or as an adjunct. For instance, in the examples (5.34) [Moro, 2017] the role of '*a prime number*' in (5.34a) is the same as in (5.34b) which is a predicate. Finally, McNally [1993] and Barwise and Cooper

[19]Theses characteristics might slightly differ from language to language (see [McNally, 2011]), however the discussion only concerns (as for the other semantic properties) English ES.

[20]Expletive: "a syllable, word, or phrase inserted to fill a vacancy (as in a sentence or a metrical line) without adding to the sense especially: a word (such as it in '*make it clear which you prefer*') that occupies the position of the subject or object of a verb in normal English word order and anticipates a subsequent word or phrase that supplies the needed meaningful content"-[Merriam-Webster's Online Dictionary, 2019]

[1981] agree with the latter argument in that '*there*' is the subject and irreplaceable, nonetheless, the pivot NP is not the predicate, it is the complement of the existential predicate and the coda phrase is either the NP-internal modifier or an adjunct.

(5.34) a. There is a prime number.
b. Seven is a prime number.
c. * A prime number is.

Milsark [1974, 1977] and Stowell [1978] theory, although met with general acceptance, states that at a certain level of semantic interpretation (logical form) '*there*' has no corresponding realization and is to be replaced by the pivot NP: *expletives replacement hypothesis* [Chomsky, 1986; Moro, 2017; Abzianidze, 2017b]. Such replacement would typically turn an ES into a copula sentence of which the pivot NP is the subject and the coda phrase is the post-copula predicate. However, coda phrases might not always be present in ES such as in (5.34a), and they could be non-intersective modifiers such as '*great*' in (5.35a). Thus, expletives replacement might pose some challenges (see (5.34c), (5.35b) and (5.35c) for examples) against constructing their MR. Due to that, and for the sake of unified treatment of ES, we follow Williams's [1984] proposal for taking '*there*' to be an irreplaceable semantically vacuous subject, the post-copula to be the predicate and the coda phrase to be the pivot NP modifier either internally (in the case of non-intersective modifiers as '*great*' in (5.35a)) or as a conjunct. Doing so, will then make us treat ES as special kinds of copulas, that eventually turns into equality between '*there*' which indicates the existence or presence of something, and the post-copula which describes the identity of that thing.

(5.35) a. There are some great tenors.
b. * Some great tenors are.
c. *Some tenors are great.

5.8 Subordinate Clauses

Beside inferring a sentence from another, in this thesis, we care to model inferences about clauses appearing within sentences, such as '*John failed to sleep*'$\models$ '*John did not sleep.*' and '*The man who loves Mary is sleeping.*' $\models$ '*a man is sleeping.*' and '*a man loves Mary.*' In English, such clauses go under the name of *subordinate clause*s (SubC). Although these clauses vary widely in form, as the examples in (5.36) show

(underlined), they generally share some common features. First, they do not stand alone as a complete sentence i.e. they have to be combined with other clause(s) to become one free standing sentences and hence they are also called *dependent clause*s. Moreover, in most cases, a SubC normally starts with a subordinate conjunction, such as '*that*' and '*if*' in (5.36e) and (5.36f), or a relative pronoun as in (5.36g). Last but not least, in their non-reduced form, SubCs always have a subject and a verb. However, each SubC has a different combination of verb-subject status. Put differently, it can be observed, again from (5.36), that some SubCs have subjects which are *overt*, while others have a *zero* subjects (having a zero subject could mean not having a subject at all, or having a *non-overt* one), along with a finite or a non-finite verb.

(5.36)

	SubC in a sentence	Verb	Subject
a.	I went to London to see the queen.	To infinitive	Zero
b.	John managed to eat a ripe peach.	To infinitive	Non-overt (John)
c.	Smith saw John sign the contract.	Bare infinitive	Overt (John)
d.	Smith saw John signing the contract.	Present Participle	Overt (John)
e.	I know that John has loved Mary.	Tensed	Overt (John)
f.	You should go to London if you want to see the queen.	Tensed	Overt (you)
g.	The man who loves Mary slept.	Tensed	Overt (who)

In this work, we focused on the SubCs of two syntactical roles; clauses modifying NPs (*relative clauses*) in particular, and others acting as complements for a range of transitive verbs called *attitude verbs* [Abzianidze, 2017b]. In their discussion, as their implementation, we stress the importance of mapping into their MRs the status of their subject-verb combination as well as their behaviour in positive and/or negative contexts in drawing the relevant inferences.

5.8.1 Relative Clauses

A relative clause (RC) is the kind of SubC that comes after an NP to refine it, as in (5.37a), or to provide extra information about it, such as in 5.37(b). The latter is called a *non-restrictive attributive* RC while the first is a *restrictive* RC and both function as modifiers. RCs are generally tensed and hence have subjects. The subject is either the relative pronoun such as in (5.37a-c) or an NP such as ('*Mary*') in (5.37d) where the relative pronoun still exists, but acts as an object instead.

(5.37) a. I saw the man who loves Mary.

b. I saw my neighbour, who loves Mary.

c. I saw the man who was sitting in the park.

d. I saw the man who Mary loves.

Now, if we want to infer, for example '*I saw a man.*' or '*a man loves Mary.*', from the compound sentence (5.37d), we first treat the main clause and the RC as two separate events (as discussed previously in ensuring each tensed verb has its subject and object (if any): '*I saw the man*' and '*Mary loves who*'. Then, we resolve the relative pronoun '*who*' to the NP it modifies (i.e.'*the man*') to get '*Mary loves the man.*'

Natural logic inference systems, discussed in Section 4.3, focused on a particular perspective of RCs (one that reflects its use in the FraCas examples) that is in the context of the conservativity of GCs. Obtaining from premises such as '*some animals are cats*' and '*all cats meow*' the compound conclusion is '*some animals are cats who meow*'. The MonaLog system relies on a specific rule to do so (see Table (4.7)) and the same goes for natural tableau. In our system, arriving at the compound sentence is achievable using the same machinery used to derive its parts. Put differently, to prove '*some animals are cats who meow.*', it is first *normalized* (see Section 7.5 and Section 7.6) into two events: '*some animals are cats*' and '*cats meow*', and then each of these events is proved separately from the given premises, which is in this case straightforward.

5.8.2 Attitude Clauses

An *attitude clause* is a clause that complements an attitude verb. These verbs, such as in: '*manage to*', '*fail to*', '*able to*', '*remembered that*', '*know that*' and '*believe that*', give rise to certain *presuppositions*[21] and/or entailments about their complements [Abzianidze, 2017b]. For example, (5.38a) presupposes that '*Mary had breakfast.*', while (5.39b) entails that '*Mary did not have breakfast.*' and presupposes that she *intended* to [Karttunen, 2012]. One semantic classification of such verbs is given in [Karttunen, 1971, 2012, 2015b], in which constructions such as '*remembered that*' are

[21]The concept of presupposition and the constructions that induce it is a well-established debate that goes way back to [Kiparsky and Kiparsky, 1970]. The examples discussed in this section are based on Karttunen's [1971; 2012; 2015b] demonstration of the properties of certain verbs: implicatives, factives, and non-factives, in the setting of natural logic and the affect of polarities on inferring complement propositions, which is what we are interested in.

called *factives* and upon their assertion the speaker, as well as the addressee, is committed to presuppose the truth of their complements. That assumption is not affected by negation, if-clause nor questions [Karttunen, 2012]. For example, (5.38b) still presupposes that '*Mary had breakfast.*' even when '*remembered that*' is negated. In contrast, a *non-factive* verb such as '*pretend*', despite the truth of the embedding sentence, holds a counter presupposition about its embedded complement (i.e. its negation) [Abzianidze, 2017b].

(5.38) a. Mary remembered that she had breakfast.

b. Mary did not remember that she had breakfast.

On the other hand, constructions such as '*remembered to*' that involve entailment (with a possibility of carrying a presupposition) about an *implicit* proposition in their complements are called *implicatives*. In contrast to presupposition, entailment is sensitive to negation, i.e. what can be entailed from an implicative complement, whether a proposition or its negation (if at all), depends on the polarity of the context it appears in. For instance, (5.39a) entails '*Mary had breakfast*' while in (5.39b) '*Mary did not have breakfast*' is entailed. This entailment property of '*remembered to*' can be described using Nairn et al.[2006] implicative signature $(+/-)$ or Karttunen's [2012] notation $(++|--)$, which both mean that if a sentence including '*remembered to*' is positively asserted, then it is the affirmative of the complement, otherwise it is the negation of the complement that is entailed.

(5.39) a. Mary remembered to have breakfast.

b. Mary did not remember to have breakfast.

In the following, the discussion about implicatives is continued. It illustrates on one side, the entailment properties of other implicative constructions. On the other, it demonstrates an essential restriction for entailing an implicative complement which is tense-agreement. After that, although in this thesis's scope, we do not provide any special treatments of presuppositions, the discussion is extended to some (non-)factive verbs, such as '*know*', '*believe*', '*doubt*' and '*see*'. We argue that implicative entailment patterns (below) can be used to explain the inference properties of those attitude verbs.

Implicatives

Nairn et al. classify implicatives according to their entailment properties into nine

implication signatures[22]. Each signature is a pair (P/N) that indicates whether an implicative verb entails the affirmative (+), negation (−), or null (0) from its complement in both positive and negative contexts. For instance, '*managed to*' has the signature (+/−) meaning that it entails its complement in positive contexts and the opposite of its complement in negative contexts. Karttunen further grouped implicatives into *two-way* and *one-way* implicatives. The latter group includes constructions that yield an entailment about their complements only under one polarity. The first are those that yield an entailment in both negative and positive contexts. Table 5.4 lists some examples of two-way and one-way implicatives[23] along with their entailment properties. Each property is one of Nairn et al.'s nine implicative signatures but in Karttunen's notation (i.e. $+P|-N$) as we found it to be much clearer to have the polarity of the context present.

Two-way implicatives		One-way implicatives			
$++\|--$	$+-\|-+$	$++\|-0$	$+-\|-0$	$+0\|--$	$+0\|-+$
manage to	fail to	cause NP to	refuse to	can	hesitate to
bother to	neglect to	force NP to	be able to	be able to	
remember to	forget to	make NP to	prevent NP from		

Table 5.4: Implicative constructions and their entailment properties [Karttunen, 2015b]

To understand why some implicatives carry an entailment about an implicit proposition under one polarity or the other, consider the examples in (5.40). According to the above table, '*be able to*' has a (−−) entailment property, which means (5.40a) entails (5.40c). However, if a person says (5.40b), where '*be able to*' appears to be in a positive context, one is inclined to take (5.40d) as a valid conclusion. Nonetheless, it is not a contradiction for the speaker to continue to say '*but she did not do it*'. That is, the sentence in (5.40d) is said to be an *implicature*[24] of(5.40b) and not an entailment.

[22]The signatures are +/+, +/−, +/0, −/+, −/−, −/0, 0/+, 0/−, and 0/0

[23]The table lists simple implicative verbs that were taken from [Karttunen, 2015b]. However, for a discussion about *phrasal implicatives* such as '*take the time to VP*', see Karttunen [2012]

[24]Implicatures are *cancellable* [Grice, 1975] [Horn, 2006], in the sense that they can be inferred only if they do not contradict other facts known to the speaker and addressee. We believe that some of Karttunen's two-way implicatives are actually one-way implicatives in a similar sense. That includes '*forgot to*' and '*fail to*', where what is entailed in a negative context can be cancelled. For example '*Mary did not forget to lock the door*', we cannot say that means '*Mary locked the door*' because she might never have intended to.

(5.40) a. She was not able to log in to my computer.

b. She was able to log in to my computer.

c. She did not log in to my computer.

d. She logged in to my computer.

Beside distinguishing their entailment patterns, Karttunen [1971] has further provided a restriction that differentiates between implicatives and other constructions (*non-implicatives*) that at first glance might be thought of as implicatives (e.g. '*hope to*', '*promise to*' and '*intend to*'). That criterion is *tense-agreement*, which is required to hold between the sentence containing the implicative construction and its complement [Karttunen, 1971], and hence the implicit proposition. So far, most of the implicative verbs we have looked at take infinitival clauses as their complements, i.e. no overt tense and thus the agreement criterion can not be observed directly. However, it is assumed that the agreement is carried by the tense of the underlying form of an infinitive clause, and that becomes apparent when time adverbials are attached [Karttunen, 1971]. To demonstrate, consider the examples in (5.41). Adding a future adverbial '*tomorrow*' to the infinitive clause of a past tensed implicative, '*manage to*' in (5.41a), has introduced the ungrammatical sentence in (5.41b). On the other hand, the infinitive complement of the non-implicative verb '*hope*' in (5.41c), can carry a future tense '*... will exercise.*' or a modal '*... might exercise.*' without problems, but not the past tense [Karttunen, 1971].

(5.41) a. John managed to exercise.

b. ∗John managed to exercise tomorrow.

c. John promised to exercise.

d. John promised to exercise tomorrow.

Another example of of a non-implicative construction, but slightly different than the above, correctis '*used to*' in the examples (5.42).It has **two** consequences: the proposition in question held before now and that it does not hold now. That cannot be handled using the pattern $++|--$, since the positive version of '*used to*' has one negative and one positive consequence, and it also breaks the tense agreement constraint.

(5.42) a. John used to beat his wife.

⊨John beat his wife.

⊨John does not beat his wife now.

b. John used to be a businessman.

⊨John was a business man.

⊨John is not a business man.

To sum up, to accurately infer an implicit proposition from an implicative complement, an inference system has to have mechanisms for ensuring that the entailed proposition obeys the implicative verb entailment pattern and has a tense-agreement with the implicative verb. Moreover, the infinitival complements of implicative verbs often have a non-overt subject, which is implicitly the same subject as the implicative verb. Therefore, resolving the subject of the complement to the subject of the implicative verb is something to consider as well. With that having been said, for our inference engine we have considered the following: (1) marking implicatives with their entailment properties during the syntactical analysis (Section 6.2.3), (2) resolving the subject of an infinitive clause to the subject of the main sentence, (3) designing inference rules to handle the different entailment patterns: $++$, $+-$, $-+$, $--$, $+0$ and -0 (see Section 8.3.1),where each rule ensures passing time information of the main utterance to the infinitive clause if required.

Tense-agreement conditions add to the general importance of modeling time information in our MR (discussed in Section 5.5.3). Among the discussed natural logic inference systems (Section 4.3), only NatLog (Section4.3.1) that has a computational consideration for passivization and morphology variation, which acknowledges the need for tense-agreement in implicative sentences. Natural tableau (Section 4.3.3), on the other hand, has no means of tense and aspect representation, which has lead to licensing several invalid inferences[25] such as inferring '*John is exercising.*' and '*John will exercise.*' from (5.41a).

Propositional and Perspective Attitudes

Propositional attitude sentences[26], as in (5.43) and (5.44), express the subject's/agent's attitude (mental state) with respect to a proposition. The attitude is conveyed by the

[25]By practical testing of some examples using the online version of their system: https://naturallogic.pro/LangPro/

[26]Also called propositional attitude reports [Swanson, 2010].

attitude verb and the proposition is commonly its complement (that-clause). Similar to implicatives, in this thesis, we are interested in modeling the entailment patterns of two kinds of attitudes: *knowing*, *believing* and *doubting*.

The verb '*know*', in (5.43), carries the speaker's (not-)knowing attitude towards the proposition '*there is a man in the park*'. In both cases, the speaker and the addressee presuppose the validity of the proposition. However, in the case of knowing (5.43a), the speaker carries a strong assertion, and possibly evidence (such as being in the park himself) that '*there is a man in the park*', i.e. it is a valid proposition. The same cannot be said about '*not know*' in (5.43b), because it implies two options: either there is indeed a man in the park, but the speaker did not have access to the park in order to acknowledge it, or there is not any man in the park in the first place. Therefore, the attitude verb '*know*' is classified in the literature as factive, and entailing to its embedded proposition only in positive declarative sentences [Karttunen, 1971; Swanson, 2010]. Hence, '*know*' has the entailment pattern $(++|-0)$.

(5.43) a. I know that there is a man in the park.

$\models$ There is a man in the park.

b. I do not know that there is a man in the park.

$\not\models$ There is a man in the park.

$\not\models$ There is not a man in the park.

On the other hand, the attitude verb '*believe*' in (5.44), is "neither factive nor entailing" [Swanson, 2010] and hence its entailing pattern is $(+0|-0)$. That is because, by choosing '*believe*', the speaker expresses uncertainty about the proposition, otherwise, the speaker would have shown a stronger attitude towards the proposition (say, for example, used '*know*' instead of '*believe*'). Another verb that has the same entailment properties as '*believe*' but carries the opposite attitude towards the same proposition is the verb '*doubt*' (notice such opposition between the examples (5.44a and b) and (5.44c and d)).

(5.44) a. I believe that Mary stole my watch.

b. I do not doubt that Mary stole my watch.

c. I do not believe that Mary stole my watch.

d. I doubt that Mary stole my watch.

The *perspective attitude* verb '*see*' shares the same factivity and entailment property as '*know*'. That is, only in a positive declarative utterance (e.g. 5.45) the embedded proposition can be entailed. In a negative context, as in (5.45b), nothing can be entailed because '*not seeing*' something happen could mean it did happen but the speaker did not see it, or it did not happen in the first place.

(5.45) a. I saw a man sign the contract.

$\models$ A man signed the contract.

b. I did not see a man sign the contract.

$\not\models$ A man signed the contract.

$\not\models$ A man did not sign the contract.

A key difference between propositional and perspective attitudes is that the latter requires tense-agreement between the time of seeing and its embedded event. However, in the first propositions are tensed, and their time might be different from the attitude verb's time.

5.9 Scope Resolution and Cooper Storage

A final semantic phenomenon to discuss in this chapter is *scope ambiguity*. Consider the example (5.46a) and its possible readings in (5.46b) and (5.46c). Such differences in reading of the sentence, are caused by the scope ambiguities of the QNPs '*every man*' and '*a woman*', where in the first reading '*every man*' has a wider-scope than '*a woman*', and in the second it is the other way around. Unfortunately, these scoping possibilities are not always syntactically reflected [Blackburn and Bos, 2005]. Moreover, a syntactical analysis of a sentence does not always correspond to the semantically more sensible or preferred reading. There exist several approaches for solving the scope ambiguity problem[27]. These approaches, as Ruys and Winter [2011] categorise them, are either syntactic or semantic[28]. Approaches of the first group (such as *Quantifier Raising* [Chomsky, 1976; May, 1978] and *Quantifying-in* [Montague, 1973] theories) rely on generating several distinct syntactical analyses for the same sentence, each of which

[27] See Blackburn and Bos [2005], Ruys and Winter [2011] and the cited references for detailed illustration of these approaches.

[28] Although, Ruys and Winter [2011] demonstrated that their categorisation is based on the majority of approaches as there exist some syntactic approaches with semantic repercussions and vice versa.

corresponds to a different semantic reading. In other words, syntactic approaches require modifying and extending the syntactic rules in order to arrive at different scoping possibilities. Semantic approaches (e.g. *Cooper Storage* [Cooper, 1983; Keller, 1988] and *Hole semantics* [Bos, 1996]), on the other hand, maintain a single (most straightforward) syntactical analysis of a sentence, but apply their semantic interpretation rules in various ways to obtain the different scope reading.

(5.46) a. Every man loves a woman.
b. *For each man there is a woman which he loves.*
c. *There is that one particular woman that is loved by every man.*

In this thesis, we use a semantic approach that bears a resemblance to the Cooper storage approach and its adapted version in [Alshawi and van Eijck, 1989; Alshawi, 1992], to handle scope ambiguities that are caused by different sources; QNPs and negation, in particular. The basic ideas of Cooper storage are as following:

- Each NL expression corresponds to a single syntactical analysis.
- From that analysis, a *core* representation for that expression is built.
- While building the core representation, the meanings of scope-bearing expressions (i.e. *binding operators*) are either applied *in-situ* to the core, or collected and kept in a *store* for later application.
- For an expression, the ordered pair $\langle Core, Store \rangle$ counts as its generalized MR. The resolution of this will result in constructing (possibly) other MRs, each of which corresponds to a scoping possibility.
- The resolution process actually involves *retrieving* binding operators and *applying* them to the core representation in a certain order. According to Cooper's assumption, a retrieved operator is combined with the respective free variable in the core expression at any point of the meaning construction process [Ruys and Winter, 2011].
- The pair $\langle Core, Store \rangle$ is regarded as an expression's semantic reading only if there are no more binding operators to apply; i.e. the store is empty.

To demonstrate these steps, let the pair Φ_0 in (5.47)[Ruys and Winter, 2011] be the generalized MR for the sentence (5.46a)[29]. The core representation is the lambda term

[29]The reader might find a slight difference in notation in regard to the representation of Cooper's Storage and related aspects, however the examples, as well as the notation used here, are based on Ruys and Winter [2011] that itself was based on Carpenter [1997].

$\textsc{loves}(x,y)$ and $\langle x/Q_1, y/Q_2\rangle$ is the binding operators list. Among the different MRs in (5.48) only Φ_3 and Φ_4 are fully resolved (their store is empty) and correspond to the semantic readings (5.46c) and (5.46d) respectively.

$$\begin{aligned}
&\Phi_0 = \langle \textsc{loves}(x,y), \langle x/Q_1, y/Q_2\rangle\rangle, \text{ where}\\
&Q_1 = \lambda A.\forall z[\textsc{man}(z) \rightarrow A(z)]\\
&Q_2 = \lambda B.\exists u[\textsc{woman}(u) \wedge B(u)]
\end{aligned} \tag{5.47}$$

$$\begin{aligned}
&\Phi_1 = \langle Q_1(\lambda x.\textsc{loves}(x,y)), \langle y/Q_2\rangle\rangle\\
&\Phi_2 = \langle Q_2(\lambda y.\textsc{loves}(x,y)), \langle x/Q_1\rangle\rangle\\
&\Phi_3 = \langle Q_1(\lambda x.Q_2(\lambda y.\textsc{loves}(x,y))), \langle \phi\rangle\rangle\\
&\Phi_4 = \langle Q_2(\lambda y.Q_1(\lambda x.\textsc{loves}(x,y))), \langle \phi\rangle\rangle
\end{aligned} \tag{5.48}$$

Similarly, following the conventions described in the Sections 5.3– 5.8 we will obtain, for a sentence, an intermediate form called the *quasi logical form* (QLF)[30]. That form is unresolved (it reflects the direct reading of the dependency tree) and has scope-bearing expression represented as binding operators. The resolution of a QLF involves involves several steps. First, the form is turned into a *main proposition* and a *stack* of all binding operators. The main proposition is an abstracted expression that is obtained by extracting the binding operators introduced by its arguments and leaving the variables they bind in their respective place. For example, consider the QLF of (5.46a) in Figure 5.4. Its main proposition `den(love)` involves a `subject` and a `dobj`. Their representations are extracted and placed in the stack, leaving behind their binding variables (`D` for `subject` and `C` for `dobj`).

After extracting all the binding operators and placing them in the stack, they are sorted in a way that reflects the desired reading, then applied to the main proposition. For this particular part, we rely on *scope scores* such that the smaller the score, the wider the scope. The choice of scores is partially based on some empirical generalizations about restrictions on scopes and reading preferences appearing in the literature (e.g. Alshawi [1992], Szabolcsi [2011] and Ruys and Winter [2011]). For instance, individual denoters, such as proper names (e.g. '*John*') are scopeless [Zimmermann, 1993]; universal QNPs cannot scope out of certain syntactic constructions such as RCs, while existentials can [Ruys and Winter, 2011], and (as discussed in Section 5.5.2) events arguments ($\exists e$) normally would have the lowest scope possible. Other scopes are for practical reasons, such as assigning scope to the utterance label `claim`, to keep them in a position we desire for further processing.

[30]The realization of the conventions described in this chapter into QLFs, and QLFs scope resolution are all described in Section 7.3 and Section 7.4 (respectively) from Chapter 7.

```
Quasi logical form
qq(claim=0,
  qq(The=1,A::{{time,[now]},A},
    qq(exists=5,B,
      {(([den(love),
        arg(dobj,qq(exists=2.0,C::{[den(woman)]@@A,C}& {card,C,=1})),
        arg(subject,qq(forall=2,D::{[den(man)]@@A,D}))]
      @@ {simple,A}),
    B)}))))
```

```
Main proposition
 {(([den(love), arg(dobj, C), arg(subject, D)]@@ {simple,A}),B)}

qq Stack
[qq(0.0, claim),
qq(5.0,{exists,B}),
qq(2.0,{forall,D::{[den(man)]@@A,D}}),
qq(2.0,{exists,C::{[den(woman)]@@A,C} & {card,C,=1}}),
qq(1.0,{The,A::{{time,[now]},A}})]
```

Figure 5.4: An example for adapting Cooper's storage for scope resolution.

5.10 Summary

Before attempting to answer the question of whether a syntactically-based MR could capture the meaning of complex semantic issues, in this chapter we have surveyed what are these issues and how they have been handled in previous systems. Then, for each semantic phenomena we have showed what representational aspects we have considered to capture their semantics in our MRs keeping in mind two important factors. The first is the ability to reason about examples involving these phenomena. The second is the ability to overcome the existing NLI systems' shortcomings (discussed in Section 4.4) with regard to certain phenomena, including defaults and propositional attitudes. Therefore, this survey is a key to our work as it is to be used for:

- Identifying (CO2.1) the semantic features that are ought to be encoded on syntactical trees—Chapter 6.
- Designing (CO2.2) the transformational staged for normalizing syntactical trees into forms that are semantically rich and suitable to used for the inference engine—Chapter 7.
- Designing (CO2.3) a number of inference rules for handling higher-order constructions: defaults and propositional attitudes—Chapter 8.

Chapter 6

Parsing and Pre-Processing

In this chapter we discuss the first part (Figure 6.1) of the inference system presented in this thesis, as illustrated in Figure 1.2 (repeated here as 6.2), which is the *dependency parser*. It parses the sentences of an inference problem into *dependency trees* (DTs). These trees are considered to be the basis for building the desired final forms: normalized DTs. The structure of the final normalized forms is critical for their use with the inference engine, so the DTs produced by the parser must be suitable for conversion to the desired form. Therefore, to ensure that we obtain DTs that conform to the syntactic and semantic requirements that facilitate their normalization, we have : 1) chosen an accessible *dependency grammar*[1] over an off-the-shelf robust *statistical parser*; and 2) identified (CO1.1) a number of semantic features that ought to be encoded on trees. After a brief introduction, we provide some illustrative examples that support our decision to use a grammar-based parser rather than a statistical dependency parser in Section 6.1. Then, in Section 6.2, the grammar used to produce our DTs is explained. Following that, the general architecture of the actual DTs is given in Section 6.3.

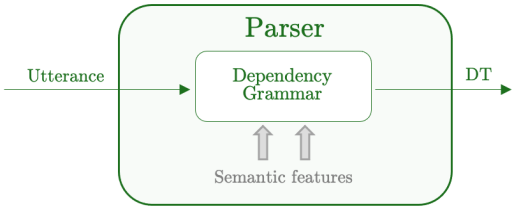


Figure 6.1: From a sentence to a DT.

[1]Implemented by my supervisor Prof. Allan Ramsay, thus the discussion and listed examples in Section 6.1 and Section 6.2 are based on Prof. Ramsay's background notes for this grammar.

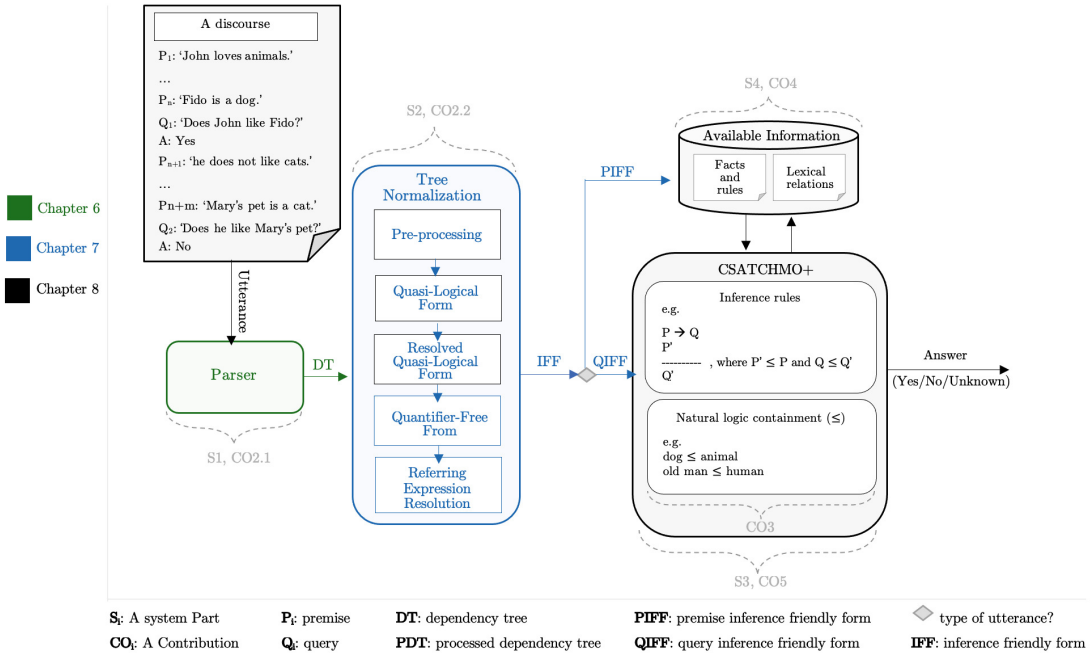


Figure 6.2: Data-flow through the inferential system.

6.1 A Grammar Over a Robust Statistical Parser

Parsing can be defined generally as "the process of structuring a linear representation in accordance to a given grammar" [Grune and Jacobs, 2008, p. 1]. A NL parser is a program that takes a NL sentence (or sequence of words) and analyses their grammatical structure (with respect to some grammar) and presents them, commonly, in the form of tree: a parse tree. One classification of parsing methods is whether a parser is a *grammar-driven* parser or a *data-driven* parser. In the first, the syntactical analysis of a sentence is assigned using pre-defined grammatical rules. In the second, a pre-annotated data set (a *corpus*) is used to learn rules that can be used to determine the most *probable* grammatical analysis of a sentence [Bod, 2008; Plank and Van Noord, 2010]. In open-domain applications, data-driven parsing has an advantage over using grammar which is robustness. However, the aim of this thesis is not wide-coverage as, e.g. NatLog and natural tableau. Instead, we are mainly interested in exploring the possibility of reasoning about deep semantic phenomena (Chapter 5) using parse trees as the basis for reasoning. For that to work, we have to be confident that the obtained trees express exactly the desired relations, and the only way for doing so is by taking control of the grammar. There are a number of desiderata to be considered when choosing a parser for our task, summarised below. We illustrate these issues by

considering the output of three well-known parsers, namely the *Stanford Dependency Parser* (SDP) [Chen and Manning, 2014; Manning et al., 2014], *EasyCCG* [Lewis and Steedman, 2014] and *MALTParser* [2][3]. We have not included examples from every publicly available parser (or even every version of these two: the Stanford Dependency Parser can generate either 'basic dependencies' or 'enhanced++ dependencies'. As expected, the latter include more detail than the former, but in some cases the trees that are derived are in fact completely different: for '*one of the leading tenors is Pavarotti*' the basic dependency version assigns '*one*' as the subject and '*tenors*' as a daughter of '*one*' while the enhanced++ version assigns '*tenors*' as the subject and '*one*' as a daughter of '*tenors*'). The aim here is simply to show what any parser that is to be used for our task must do.

6.1.1 Semantic Interpretability

The most crucial requirement is that the trees the parser produces make sense when we come to use them for inference. Many parsers, e.g. EasyCCG, for instance, assign the modifier '*in the park*' from '*the man in the park saw me*' as a daughter of the NP '*the man*' rather than as a daughter of the NN '*man*'. This means that PPs are treated differently from other modifiers such as adjectives, which makes it very difficult to write rules for handling them. There are numerous other examples of trees that are unsuitable for reasoning, e.g. the decision in SDP to make the predicative noun '*fool*' rather than the copula '*is*' the head in '*John is a fool*' (this also violates the requirement of consistency below, since it means that '*an Italian became the world's greatest tenor.*' and '*an Italian was the world's greatest tenor.*' have entirely different trees).

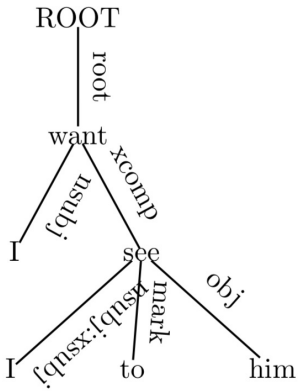
6.1.2 Sensitivity

A corollary of the first requirement is that the parser must assign distinct trees to sentences where similar phrases play different roles. Both SDP and EasyCCG, for instance, assign identical structures to (6.1) and (6.2) (see Figure 6.3), making it impossible to infer that in (6.1) my reason for going was to see him whereas in (6.2) seeing him was the thing I wanted.

[2]We are using version 4.20 of SDP and the default version of MALTParser provided in the NLTK, i.e. using the Wall Street corpus for training and Matthew Honnibal's Greedy Averaged Perceptron tagger.

[3]We have included a number of trees produced by SDP and MALTParser for illustration. The trees produced by EasyCCG are very hard to interpret, and take up a great deal of space, and we have therefore not included any from this parser.

(6.1) I want to see him.



(6.2) I went to see him.

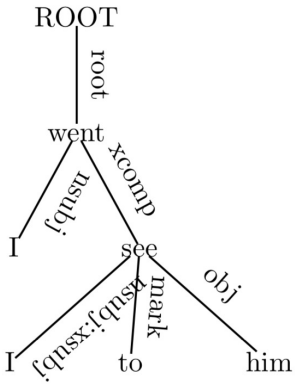
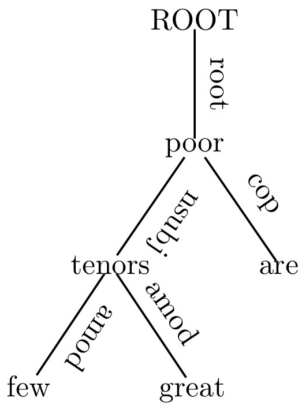


Figure 6.3: SDP trees (MALTParser trees are almost identical).

6.1.3 Consistency

Similarly, the parser should assign similar trees to closely related sentences such as declarative and interrogative forms of the same proposition. EasyCCG, for instance, assigns entirely different trees to (i) '*there are great tenors who are Swedish.*' and (ii) '*are there tenors who are Swedish?*', which means that it would be impossible to infer that the answer to the question (ii) was '*yes*' from the premise (i). SDP assigns entirely different structures (see Figure 6.4) to '*few great tenors are poor*' and '*most great tenors are rich*', which again makes it impossible to infer any relation between these two. Similarly MALTParser produces different trees for (6.5) and (6.6) in Figure 6.5, with '*there*' treated as an expletive in (6.5) and as an adverbial modifier in (6.6).

(6.3) Few great tenors are poor.



(6.4) Most great tenors are rich.

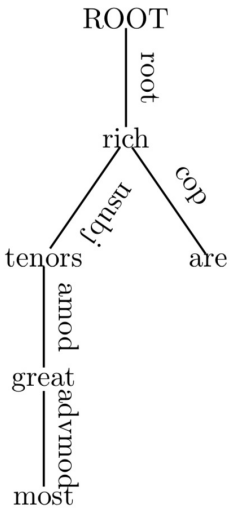
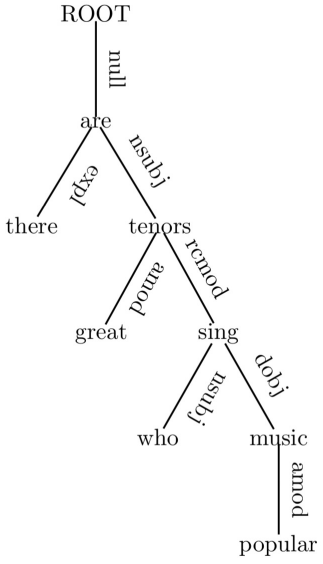


Figure 6.4: SDP trees for '*few great tenors are poor*' and '*most great tenors are rich*'.

(6.5) There are great tenors who sing popular music.



(6.6) Are there great tenors who sing popular music?

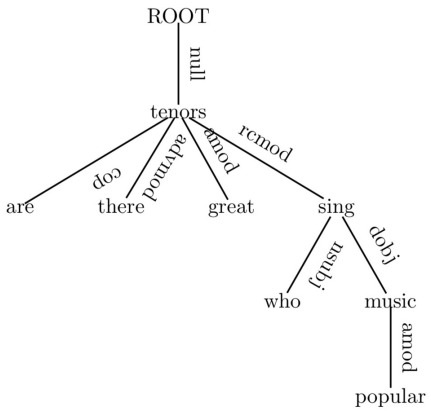


Figure 6.5: MALTParser trees for '*there are great tenors who sing popular music.*' and '*are there great tenors who sing popular music?*'

Finally, the decision to make the predicative noun the head of a copula sentence also obscures the close relationship between e.g. '*One of the leading tenors is Pavarotti*' and '*Pavarotti is one of the leading tenors*' (see Figure 6.6).

(6.7) One of the great tenors is Pavarotti.

(6.8) Pavarotti is one of the great tenors.

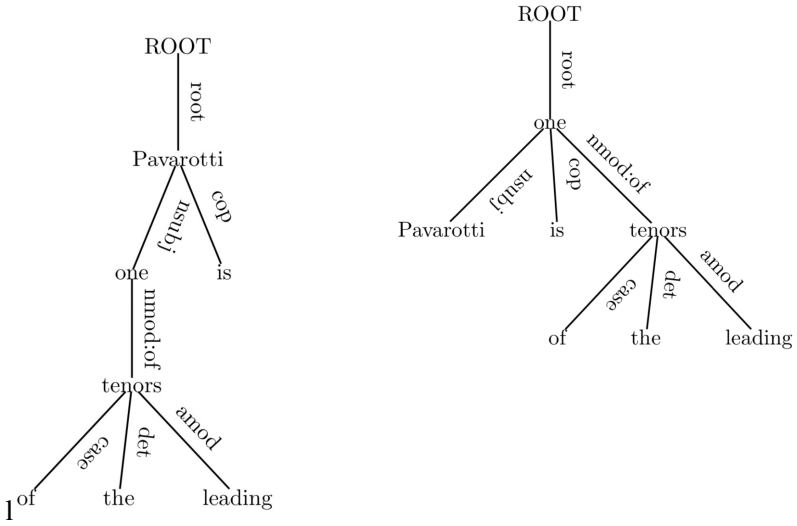


Figure 6.6: SDP trees for '*One of the great tenors is Pavarotti*' and '*Pavarotti is one of the great tenors*' (MALTParser trees are almost identical).

6.1.4 Non-reentrancy

The structures assigned by the parser **must** be trees. SDP, for instance, assigns two heads to '*man*' in '*I saw the man who Mary said she wanted to meet*', namely as the object of '*saw*' and the object of '*said*', and it also assigns '*said*' as a daughter of '*man*'. If the structures assigned by the parser are not trees then tree-matching algorithms such as those deployed in natural logic approaches cannot be used.

6.1.5 Informativeness

Any information that is not included in the parser output will be unavailable for inference. Both EasyCCG and SDP, for instance, omit anything about tense/aspect or about number. That means that using trees produced by these parsers would make it impossible to distinguish between (i-a) '*John loves Mary*' and (i-b) '*John loved Mary*',

or between (ii-a) '*Some fool has left the door open*' and (iib) '*Some fools have left the door open*'. (i-a) and (i-b) would be mutually entailing, and likewise (ii-a) and (ii-b).

Some of this information can be fairly easily recovered, e.g. SDP can be reconfigured to output fine-grained part-of-speech tags that do contain tense and number information. Some cannot. In particular, dependency parsers typically have difficulty with traces/WH-items. EasyCCG, for instance, simply fails to say anything about the role of '*who*' in (i) '*I saw the man who she expected to marry*' and (ii) '*I saw the man who she expected to marry her*'. SDP again deals with this issue by assigning multiple heads to '*man*', making it a daughter of '*saw*', '*marry*', '*expected*' and '*who*', again making it impossible to use tree-matching algorithms for comparing sentences.

SDP's treatment of auxiliaries also loses crucial information, since it simply assigns all the auxiliaries in a complex sequence like '*John would have been singing in the choir today*' as direct daughters of '*singing*', thus making it impossible to trace the route between speech time and event time. This information **is** recoverable, but only by paying attention to the left-to-right order of the items in the tree, violating the notion that all that matters is the relations between heads and their daughters.

(6.9) John would have been singing in the choir today.

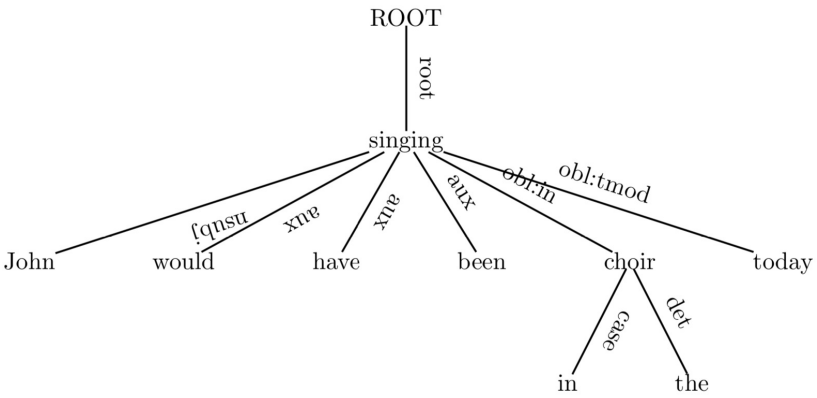


Figure 6.7: Flat structure for auxiliary sequences in SDP tree (MALTParser tree is almost identical).

6.1.6 Transparency

The output of any parser is determined by the underlying grammar. This is obviously true for grammar-based parsers, but it also holds for parsers obtained by running

a machine-learning algorithm over a treebank. The major difference is that with a grammar-based parser it is possible to ascertain which rule(s) gave rise to a particular analysis, and hence to work out how to patch any major issues. The grammar behind a treebank-based parser is implicit in the guidelines given to the annotators (the similarity between the outputs of SDP and MALTParser suggest that guidelines given to the annotators for their training data, i.e. the implicit grammars behind these two parsers, were very similar). This makes it impossible to find out why a parser has assigned a given analysis, and hence impossible to fix things that are wrong.

To sum up, the point here is not that EasyCCG or SDP or MALTParser in particular are unsuited to our task. These are typical issues that are likely to arise with any externally supplied parser. If we do not have control over the underlying grammar then the parser is likely to provide analyses that are incompatible with our requirements; if we do not have control over the labels that it provides then it is likely to omit information that we need; if it provides identical analyses for sentences that are clearly different, or distinct analyses for sentences that are clearly closely related, we will be unable to carry out the requisite inferences; and if the analyses are, in fact, lattices rather than trees then we will be unable to use tree-matching algorithms.

Rather than wandering through all the publicly available parsers [4], or even all the versions of a single parser, trying to find something suitable, we therefore prefer to use a grammar-based parser where we can ensure that the trees that we obtain do conform to our requirements.

6.2 The Used Dependency Grammar

The term *dependency grammar*,does not refer to a particular grammar, but to a family of grammar formalisms in which syntactic structures are described in a certain way [Nivre, 2005]. That is, words are linked via grammatical relations (e.g. subject, direct object, adjective, etc.), called *dependency relations* or *dependencies*. These dependencies are binary and directional, thus are often represented with labelled arcs ($H \xrightarrow{relation} D$) from a head to a dependent (or a daughter as we tend to use in this thesis) [Nivre, 2005]. These arcs collectively form a rooted tree which is the dependency tree [Nivre, 2005].

The grammar we are using generates DTs. However, the rules of combination (given below in the definition of signs) actually relate words and trees, where a tree

[4] Some of which, e.g. SyntaxNet, require considerable effort to install.

consists of a head word and a (possibly empty) set of subtrees. In particular, the description of the item at the head of a tree may include information about the nature of the daughters. It is important, for instance, to know whether any of the daughters of a verb are *WH-marked*, in order to be able to spot that the verb can be used as a relative clause (e.g. that '*loves*' in '*the man who she loves has gone to Mexico*' has the relative pronoun '*who*' as a daughter: the word '*loves*' carries no information about whether '*loves*' can be a modifier of '*man*', whereas the tree `[loves, [she], [who]]` can be seen to **contain** a relative pronoun and hence can be used as a modifier). This is a rather fine distinction, and is often blurred in discussions of dependency grammar, but it is important to keep it in mind. Therefore, the grammar we are using is not strictly a dependency grammar, but a simplified version of the *Head-driven phrase structure grammar* (HPSG) [Pollard and Sag, 1987, 1994] in which the head of a tree is not strictly a word but a typed *feature structure* called a *sign*. In a DT, words always have sets of daughters. However, words also occur in isolation, which is the case of heads and empty sets of daughters. For the sake of uniformity, we use the term *sign* to mean either a word or a tree. To describe a sign, we specify its features. There is a finite set of things one might want to say about such an entity (e.g. things about its meaning, its position within the sentence where it appears, or about some property that governs what other signs it can be related to) and these can be organised into groups in a HPSG-like complex feature structure where features can have sub-features and those sub-features can have their own sub-features. A sample of a sign is given in Figure 6.8, in which features are represented as complex terms where the functor (e.g. `sign`) is the label of the feature and its arguments (e.g. `(structure(_), syntaxt(_), morphology(_),semantix(_),...)`) are the sub-features which might have sub-features themselves (indicated by the list brackets: `(_)`). One important feature (which we referred to as our combining rule above) of a `sign` is its `args([_])`[5] as it is the place to name other `sign`s (if any) as the daughters (see, for example, the `sign` of the transitive verb '*eat*' given in Figure 6.9).

Although the grammar we are using came with an extensive list of features a sign might have, in this research we have identified (CO1.1) a number of other features which carry semantically important information and hence are important to get encoded on trees and the parser's developer, Allan Ramsay, has extended the parser to

[5]Note that we use a different brackets for `arge` and that is because (in comparison to listing the sub-features of a feature) we list other signs inside the feature structure of a sign.

```
signature(
 sign(
  structure(position(moved(_), start(_), end(_), ...], index(_),language(),...)
  syntax(args([_]),
         head(cat(_),
                hd(_),
                agree(person(_),number(_),collective(_), gender(_), mass(_)),
                pronominal(_),
                nform(case(_), date(_)),
                vform(tense(_),tenselist(_),finite(_),...),
                aform((degree(_)),
                predicative(_),factive(_)),
                spec(specified(_), def(_), specifier(_), numeric(_), counted(_)),
                nonlocal(wh(_), shifted(_), zerodtr(_)),
                mod(target(_), result(_), modifiable(_))),
  morphology(_),
  semantics(content(_),
                theta(_),
                modifier(_),
                definition(_),
                polarity(_),
                arity(_),
                type(_),
                denotes(_)),
  externalviews(_)
        ...)).
```

Figure 6.8: An abbreviated version of signs' possible features.

include them. Some of the features are about bits of information that are widely recognized to be semantically relevant, but were missing from the output of the original parser and they are centred around:

- The cardinality information of generalized quantifiers–Section 5.2.
- The shape of time and aspect information –Section 5.5.3.
- Assigning scope scores for scope bearing tree parts.

Others are concerned with information that emerged from the analysis given in Chapter 5. These include:

- Assigning adjectives to appropriate semantic classes (subsectives, intersective, etc.)–Section 5.4.
- The entailment patterns of attitude verbs–Section 5.8.2.
- A way to display polarity marking.

The discussion of the information signs include can get rather complex and intricate. Therefore, in the remaining of this chapter we will explain the critical features as they arise.

```
sign(syntax(
 args([
   sign(structure(dir(position(before(-), after(+)))),
        syntax(args([]),
               head(cat(xbar(v(-), n(+))),
               nform(case(*(obj)), date(-))),
               spec(specified(+))),
        semantics(theta(dobj))),
   sign(structure(dir(position(before(+), after(-)))),
        syntax(args([]),
               head(cat(xbar(v(-), n(+))),
               nform(case(*(subj)))),
               spec(specified(+))),
        semantics(theta(subject)))])))
...
```

Figure 6.9: The `sign` for the transitive verb '*eat*'.

6.2.1 A Theory of Syntactic Categories

There are a number of broad classes of word – nouns, verbs, adjectives, adverbs, prepositions. Some of these classes share some characteristics. Adjectives, for instance, are in some ways treated as nouns (e.g. '*empty*' and '*tea*' are both noun modifiers in '*an empty pot*' and '*a tea pot*') and as verbs in some others (e.g. '*confident*' and '*sleeping*' are both modified by the adverb '*quietly*' in '*a quietly confident man*' and '*a quietly sleeping man*'). We capture this by stating that every word has a `cat`egory, and that for some words the `cat` has two features, `v`erb-like and `n`oun-like as follows:

```
%% A noun: is not verb-like, but noun-like
sign(syntax(head(cat(xbar(v(-), n(+))))))
```

```
%% A verb: is verb-like, but not noun-like
sign(syntax(head(cat(xbar(v(+), n(-))))))
```

```
%% An adjective: is a bit like a verb and a bit like a noun
sign(syntax(head(cat(xbar(v(+), n(+))))))
```

This notion is often pushed a bit further by noting that there are noun-like words and noun-like groups of words (nouns like '*cat*' and noun phrases like '*the cat*'), and verb-like words and verb-like groups of words (verbs like '*eats*', verb phrases like '*eats dried cat food*', sentences like '*the cat eats dried cat food*'). *X-bar theory* [Chomsky et al., 1970; Jackendoff, 1977] makes use of the notion of *levels*, where a verb is bar level 0, a verb phrase is bar level 1, and a sentence is bar level 2 (the notion is slightly less clear for nouns – a noun is bar level 0, a NP is bar level 2, but it's not clear whether a modified noun like '*annoying cat*' is level 0 or level 1).

Again, we follow HPSG, among other theories, in taking a slightly different approach [Müller, 2014]. Words like nouns and verbs denote descriptions of things. When combined with other appropriate words they denote something like instances of those things. For example, '*cat*' denotes the property of being a cat, '*the cat*' says that we are talking about a specific cat. We say that words such as '*the*' are *specifiers* (actually '*the*' is a Det, which is a subclass of the wider set of specifiers), and that when a noun or a verb has combined with a specifier it is *specified*.

```
cat:
sign(syntax(head(cat(xbar(v(-), n(+)))), spec(specified(-))))

The cat:
sign(syntax(head(cat(xbar(v(-), n(+)))), spec(specified(+))))
```

Other things that a word might combine with are words that add more details to its description (*modifier*s, Section 6.2.2) and others that complete its meaning (*argument*s, Section 6.2.3).

6.2.2 Modifiers and specifiers

Some words add information to a target word without fundamentally changing its meaning. This can happen in two ways:

- An unspecified word, which will typically be a descriptor of some kind (e.g. '*man*', which describes an entity as being man-like, or '*eat*', which describes an event as being some kind of ingestion of food), can be refined by a modifier ('*fat man*', '*eat quietly*')
- An unspecified word, together with any modifiers, can become specified – '*a fat man*', '*the girl with red hair*', ...

What a specifier actually does is not all that clear in the literature. However, we take it that it is a semantic notion, where a noun plus a number of modifiers (adjectives, prepositions, etc.) add some details to the description of a kind of entity. Then, in some sense, the specifier states whether what has been described is a new entity of that kind and has not been mentioned before (e.g. '*a man*'), is something that has been mentioned (e.g. '*the man*'), is about all things of that kind (e.g. '*every man*'), etc.

However, the distinction is not always that clear as some words can act as modifiers **and** specifiers at the same time, e.g. '*two*' in '*two men*' adds information about the cardinality of the set of men under discussion and specifies it as being indefinite. We therefore treat both modifiers and specifiers in very much the same way, by saying that they have a target (the tree to which they are being attached) and a result (the tree that results from attaching them). Hence, pure modifiers such as '*fat*' and '*quietly*' attach to trees that are `specified(-)` and produce results that are also `specified(-)`, whereas specifiers such as '*a*' and '*the*' attach to trees that are `specified(-)` and produce ones that are `specified(+)`. As for modifiers that carry the possibility of being specifiers, we treat them as items that supply the two bits of information mentioned above: an addition to the description denoted by its target (e.g. in '*many cats*', '*cats*' states that a set of feline animals is being discussed, and the adjective '*many*' says that this set has quite a large number of members) and a specifier to its target (e.g. '*many*' as a specifier, states the introduction of a set of cats into the conversation, similar to the determiner '*some*'). However, the latter is needed only if the target is to be used as an argument, and there is no other item acting as the specifier. Put differently, in '*John saw the many cats*' for example, '*the many cats*' is the argument of seeing, however '*the*' is the specifier and it specifies definiteness for the whole NP and '*many*' is needed only for adding on the cardinality information to the description of '*the cats*'.

The information that '*many*' supplies is reflected in its sign (see Figure 6.10). It says that: '*many*' has a target that: comes after it, is noun-like, and not specified (`specified(-)`), while the `result` of combining the target with '*many*' will be definite (`def(+)`), noun-like, and have `*(num(many)=2)` as its specifier.

So far, the discussion of specifiers has been centred on Dets (including articles: '*a/an*' and '*the*', and quantifiers such '*many*' and '*some*') as their source. However, in addition to Dets we consider specifiers to be supplied by other sources. One source is the inflectional affixes as in bare plurals (e.g.'*John eats peach**es***'). Another source is tense markings on verbs. Following the discussion of Section 5.5.3, events are situated in time and so their temporal information is supplied as specifier using the tense

```
sign(syntax(args([]),
            spec(def(+)),
            (target(sign(structure(dir(position(before(-),after(+))),
                        syntax(head(cat(xbar(v(-), n(+))),
                               spec(specified(-), def(+)),
            mod result(sign(structure(language(english)),
                           syntax(head(cat(xbar(v(-), n(+))),
                                  spec(def(+),specifier(*(num(many)=2))),
                                   ...
```

Figure 6.10: The sign of '*many*'.

markings (inflections plus auxiliary sequence) on the verb as a source. In addition, definite NPs carry definiteness specifiers either explicitly (e.g the determiner '*the*' in '*the cat*') or implicitly as in the case of proper names. A further discussion on these specifiers and their structures, along some other miscellanies, is given in Section 6.3. What we want to emphasise at this point is that specifiers introducing items will have their specifiers included as part of their definition in the dictionary.

Another semantic feature that is identified and marked is intersective/subsective signs (see Section 5.4). Take for example, the entries for the pure adjectives '*old*' and '*Italian*'. The latter is marked with the intersectivity sign `*(+)` while the other with `*(-)` indicating a subsective adjective.

```
'old'
sign(syntax(args([]),
            head(cat(xbar(v(+), n(+))), class(*(-))),
            (target(sign(structure(language(english)),
                        syntax(...)),
            mod result(sign(structure(language(english)),
                           syntax(...)))))
                                   ...
```

```
'Italian'
sign(syntax(args([]),
            head(cat(xbar(v(+), n(+))), class(*(+))),
            (target(sign(structure(language(english)),
                        syntax(...)),
            mod result(sign(structure(language(english)),
                           syntax(...)))))
                                   ...
```

Figure 6.11: The signs for '*old*' and '*Italian*'.

6.2.3 Arguments, unsaturated items, canonical order

Some words express incomplete ideas and require the presence of other words in order to make those ideas complete. The archetypal examples are verbs, which nearly always require a subject and often some other items in order to describe a state or event. We call the words that they require their *argument*s, or sometimes their *complement*s. If a word has all of its arguments it is called *saturated*, otherwise it is *unsaturated*.

In any given language, there is a canonical order for the various arguments. In English, the canonical order is SVO, in Arabic it's VSO, etc. Such order is determined in the description of the word by stating the direction of each argument. For example, see the description of '*eat*' in Figure 6.9 (restated here as Figure 6.12). It says that '*eat*' takes two arguments which are both specified NPs (`cat(xbar(v(-), n(+)), specified(+)`). However, the first is marked as its object that follows it, while the second is its subject that precedes it. The order in which the arguments appear in the list specifies the order in which they should be found, the `dir(position(before(-), after(+)))`[6] and `dir(position(before(+), after(-)))` say where one should expect to find them. We sometimes write $args = [\overrightarrow{NP}, \overleftarrow{NP}]$ as an informal abbreviation for this.

```
sign(syntax(
 args([
   sign(structure(dir(position(before(-), after(+)))),
        syntax(args([]),
               head(cat(xbar(v(-), n(+))),
               nform(case(*(obj)), date(-))),
               spec(specified(+))),
        semantics(theta(dobj))),
   sign(structure(dir(position(before(+), after(-)))),
        syntax(args([]),
               head(cat(xbar(v(-), n(+))),
               nform(case(*(subj)))),
               spec(specified(+))),
        semantics(theta(subject)))])))
...
```

Figure 6.12: The `sign` for the transitive verb '*eat*'.

Other than arguments' direction with respect to a verb and their grammatical role,

[6] Having `before` and `after` as separate features allows us to specify independently whether an argument must, can or must not appear before or after its head.

we take a particular interest in a semantic feature of a particular kind of verbs: the entailment pattern of attitude verbs (Section 5.8.2). Similar to specifiers and intersective sign, the dictionary entries for an attitude verb will include such a feature and others that in turn will feed the `sign` of the verb with the necessary values. The patterns we marked attitude verbs with embody more information than the patterns explained in Section 5.8.2. Put differently; our entailment patterns have the following structure:

$$\underbrace{[\ \underbrace{\texttt{PP / EP}}_{\text{positive context}}\ ,\ \underbrace{\texttt{PN / EN}}_{\text{negative context}}\ ]}$$

The first element (`PP / EP`) of the above pattern is about what happens when the complement of an attitude verb is inside a positive context, and the second (`PN / EN`) is about what happens when it occurs inside a negative context. The first parts of each of these, `PP` (`P`olarity in `P`ositive context) and `PN` (`P`olarity in `N`egative context) are about what polarity this complement should propagate to its daughters in a positive and negative contexts. The second, `EP` (`E`ntailment in `P`ositive context) and `EN` (`E`ntailment in `N`egative context), is about what can be inferred from that complement: its implicit proposition, the negation of its implicit proposition, or nothing. For instance, the verb '*manage*' (according to its entries in Figure 6.13) is a verb that takes a sentential complement that has a finite verb as its head, and has `[1/1, -1/ -1]` as its entailment pattern. This means that in a positive context '*manage*' marks its complement with positive mark (`1`) and entails the affirmative of its implicit proposition, while in a negative context it marks the complement with (`-1`) and entails the negation of the proposition. The verb '*fail*', on the other hand, is `[-1/-1, 0/0]` – it entails the negation of its complement in positive contexts ('*He failed to complete the final question*' entails '*He did not complete the final question*') but it doesn't entail its complement in negative ones – you can't get from '*He did not fail to complete the final question*' to '*He completed the final question*' because the reason that he did not complete it may have been that he did not even attempt it.

6.2.4 Movement

Words that need arguments specify where they expect to find them. Modifiers specify where they expect to find their targets[7]. But things do not have to be where they are

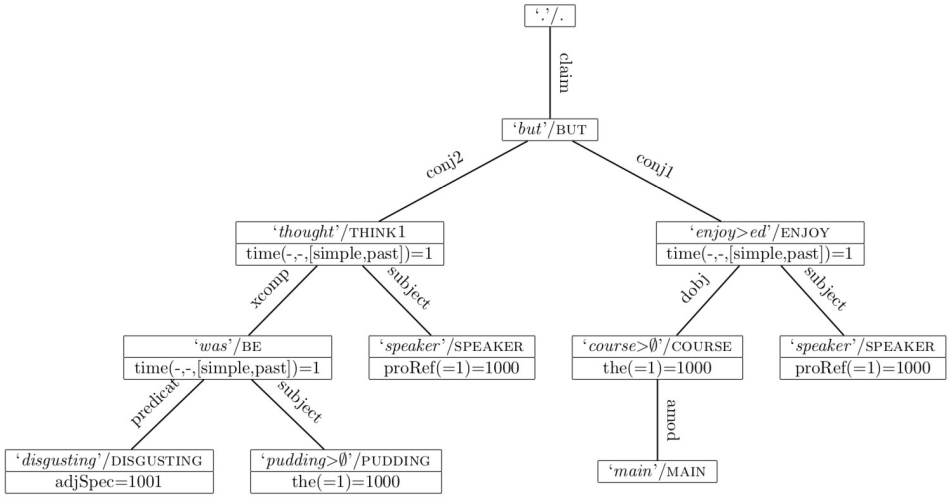
[7]The rules governing this can be quite complex, e.g. in English head final modifiers precede their targets ('*sleeping man*', '*very fat man*', others follow them ('*man sleeping quietly*', '*man with a big nose*'.

```
sign(syntax(
 args([
   sign(structure(dir(position(before(-), after(+)))),
        syntax(args([]),
               head(cat(xbar(v(+), n(-))),
               vform(finite(to),
               spec(specifier(*(time(-,identity,[simple],B)=1)))),
        semantics(theta(xcomp([1/1, -1/-1])))),
   sign(...)
 head(cat(xbar(v(+), n(-))),...
```

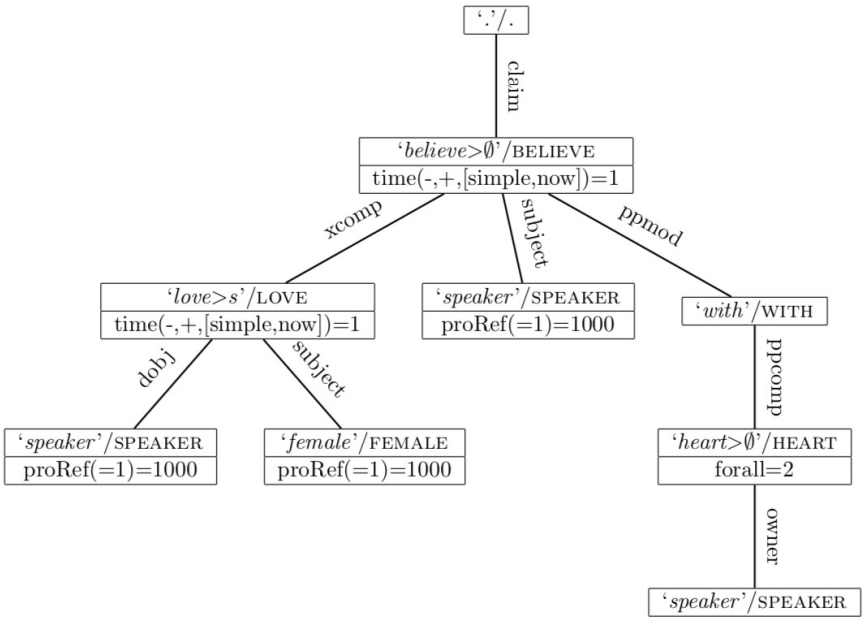
Figure 6.13: The sign for '*manage*'.

expected. There are rules that govern whether an item can, or indeed must, be shifted, and if so where it can be shifted to. The most obvious of these concerns WH-marked items, where the rule is that no WH-marked daughter of a verb may follow a non-WH-marked one (normally stated as WH-marked must be shifted to the start, but if you have multiple WH-marked daughters then they do not all have to be shifted: '*Who wants what for dinner?*'). Things get moved around quite a lot in English for emphasis (6.10) and ambiguity reduction (6.11):

(6.10) I enjoyed the main course but the pudding I thought was disgusting.



(6.11) I believe with all my heart that she loves me.

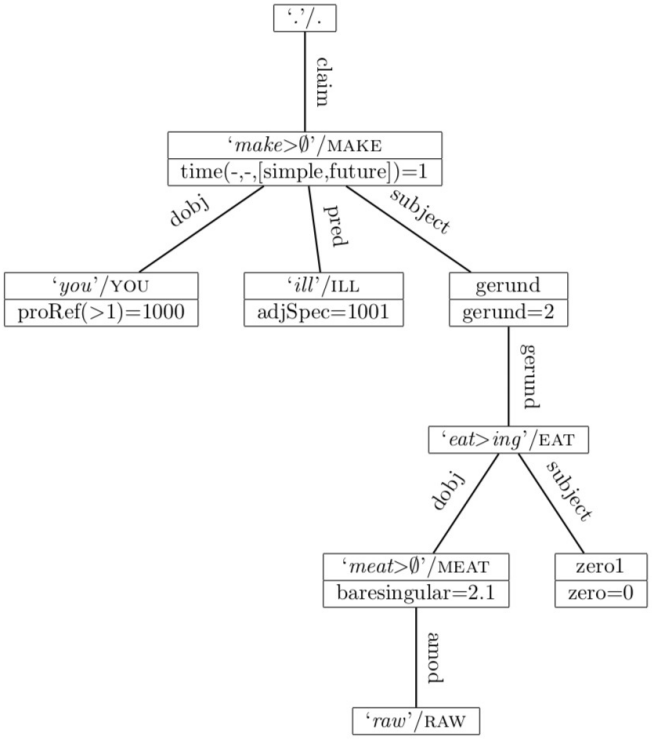


The rules that govern what can move to where are fairly intricate: they are expressed using the feature `moved`, which has polar-valued sub-features `before` and `after`, where for instance, `moved(position(before(+), after(-)))` says that the item in question has been moved to a position earlier than where one would expect it to be. As before, having separate features `before` and `after` allows us considerable freedom to say whether something may, must or must not be moved in either direction.

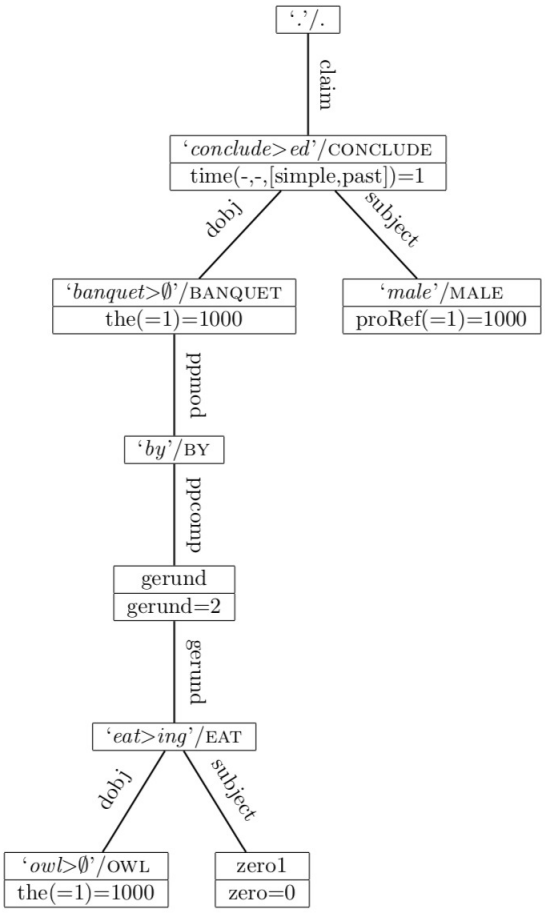
6.2.5 Internal and external views

Saying, for instance, that something is a NP conveys one of two perceptions: that it looks like a NP (e.g. it is made out of a Det followed by a noun), or that it can be used in places where one would normally use a NP (e.g. as the subject of a verb). But, it can happen that something that does not look like a NP can be used in a position where one actually expected to see a NP: '*make*' normally takes an NP as its subject, '*by*' usually takes a NP as its complement. But in (6.12) and (6.13) they combine with a present participle verb phrase (VP):

(6.12) Eating raw meat will make you ill.



(6.13) He concluded the banquet by eating the owl.



The key here is that items that look like present participle VPs can be used as NPs: internally they are VPs, while externally they are NPs. We allow items to have a range of external views: when we need to use an external view, we mark it with an extra node in the tree.

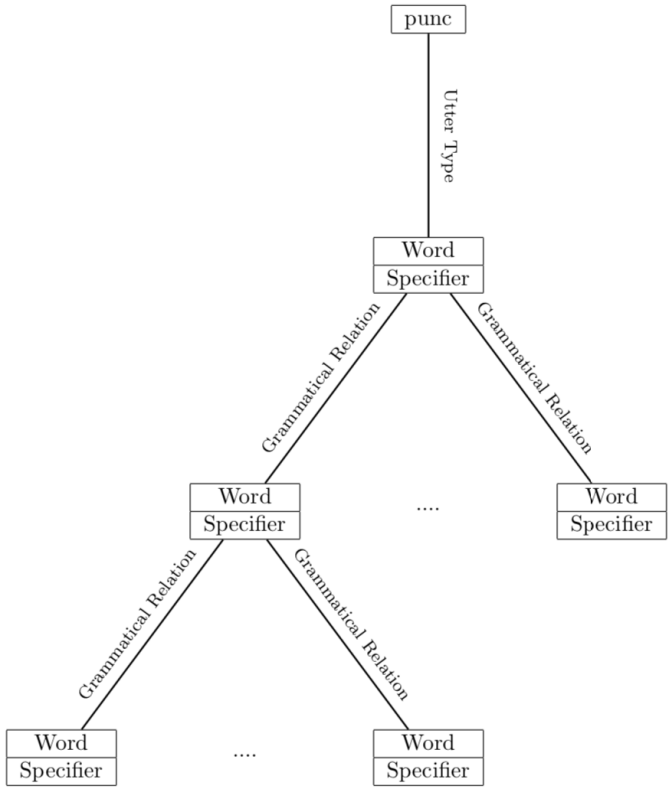
6.3 DTs' Main Structure

Now that we have seen the underlying grammatical aspects that collectively contribute to the building of DTs, this section provides an outer-look at DTs' general structure and the possible values of their parts (argument, modifiers and specifiers).

Consider the DTs' graphical representation and their equivalent data structure given in Figure 6.14. In terms of the general hierarchy, a DT of a NL utterance starts with a punctuation mark as a key indicator of the type of utterance that a tree represents. The systems (Figure 6.2) accepts two types of utterances, premise(s) which are meaningful sentences ending with '.' and are *claim*(ed) to be true, while questions are those that end with a '*?*' and are the things we *query* (questions do not have to involve aux-inversion or WH-marked subjects – any sentences ending with a '*?*' are deemed to be questions). After the utterance punctuation mark, the actual tree head node and its sub-tree will follow.

The sub-tree of any node, as Figure 6.14(b) shows, is presented as a nested list structure that consists of a `Head` (which is the node itself) and `N Daughters` (which are a list of `N` *argument(s)* and/or *modifier(s)*—`arg/mod(s)` for short), where `N` can be any non-negative integer, and in case `N` is `0` the node is called a leaf. Finally, links between nodes and their daughters are generally labelled with grammatical relations.

A word is represented in two forms: its inflected form, `word(Root_word>Marker)`, as it appears in the utterance, and its denotation `den(Root_word)`. In the inflected form, the `Marker` (if present), is actually an inflectional affix that carries a grammatical property such as tense, plurality, adjectives, etc. For example, in Table 6.1 the word '*loves*' has the form `word(love>s)`, where `>s` is the 3rd person singular present marker, and in `word(car>s)`, `>s` is the plural marker. In case of a word being the root itself, its inflected form will have the `>0` marker; `word(Root_word>0)`, indicating no inflectional affixes being attached. In addition to cases where a zero inflectional marker is added, some words might not have markers attached to them at all. This is the case for irregular words, such as `word(won)`. Words' denotations, on the other hand, are the roots



(a)

A sub-tree

```
[word(Root_word>Marker)/den(Root_word)| N arg/mod(s) sub-tree(s)]
 Head                                  Daughters
```

argument/modifier

```
arg(Gram_Label,*(Specifier_name=Scope_score),Time_var,
       [Word| N arg/mod(s) sub-tree(s)])

modifier(Gram_Label, Sem_class,
       [Word| N arg/mod(s) sub-tree(s)])
```

(b)

Instance: Utterance tree

```
[word(Punc)/den(Punc) , arg(Utter_type,
                          *(Specifier_name=Scope_score),
                          Time_var,
                          [Head Word|N arg/mod(s) sub-tree(s)]]
```

(c)

Figure 6.14: Graphical representation of a DT and the structure of the underlying representation.

without any markers for regular words. For irregular words, the denotation is the infinitive form for verbs and the singular form for nouns (e.g. the denotation of '*won*' is `win` and from '*men*' is `man`).

Type of Marker	Example
Tense	'*loves*' → `word(love>s)/den(love)` '*love*' → `word(love>0)/den(love)` '*won*' → `word(won)/den(win)`
Plurality	'*cars*' → `word(car>s)/den(car)` '*peach*' → `word(peach>0)/den(peach)` '*men*' → `word(men)/den(man)`
Comparatives and Superlatives	'*taller*' →`word(tall>er)/den(tall)` '*tallest*' → `word(tall>est)/den(tall)` '*better*' → `word(better)/den(good)` '*best*' → `word(best)/den(good)`
Adjectives and Adverbs	'*really*' →`word(real>ly)/den(real)`

Table 6.1: Examples of words' markers and their types.

A non-leaf node can have one or more `arg/mod(s)`. `args`s and `modifier`s are terms with number of parts. They each have a label (`Gram_label`) for the grammatical relation that constitutes with its head and a sub-trees of its own daughters (if any). Argument and modifier labels considered in this thesis are listed in Table 6.2 and Table 6.3 respectively. Note that an utterance, as can be seen in Figure 6.14(c), is itself an `arg` of the punctuation mark, hence, is labeled with either `claim` or `query`. Looking back at Figure 6.14 (b), it can be noticed that `modifier`s have an extra field called `Sem_class` that is a place holder for the sign indicating a modifier's semantic class: whether it it intersective, subsective, or something else.

Arguments also have two more place holders, one is for the specifier and the other for the *time variable*. We have seen in Section 5.5 that events are tensed and thus situated in time. Their constituents are normally tenseless items. Nonetheless, it is believed that their interpretation may be time-dependent [Dalrymple, 1988]. Take for example, '*John ate a ripe peach.*'. The single ripe peach that was eaten by John is a specific one that existed at that specific past time (when the event took place) and it no longer exists. Therefore, the place holder `Time_var` is used to ensure that `arg`s share the same time as the events that involve them.

As for specifiers, they have the form `*(Specifier_name=Scope_score)`. Table 6.4 provides a list of possible specifiers' names and their categories. Most specifiers' names reflect two pieces of information: what kind of specifier it is and the cardinality

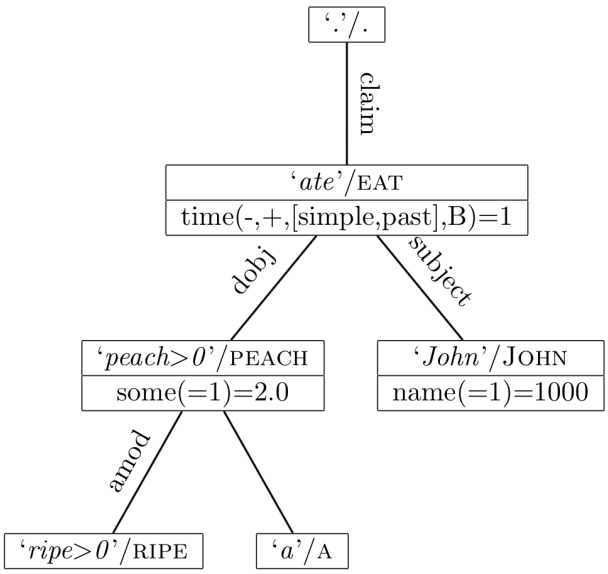
Argument label	Meaning	Notes
`claim/query`	top level markers	
`subject`	subject	
`dobj`	direct object	
`predication(PRED)`	complement of copula	the value of PRED is from Table 6.3 and depends on the type of the complement, which could be, as seen in Section 5.6, an NP, AP, or PP
`therepred`	a complement of an ES	
`ppcomp`	preposition complement	
`owner`	possession	
`xcomp([PP/EP,PN,EN])`	Attitude clauses (Section 5.8.2)	`[PP/EP,PN,EN]` constitute an entailment pattern as seen in Section 6.2.3)
`whclause`	Relative clause	used when a WH-clause is the argument of some head word
`negComp`	argument of a negation marker	

Table 6.2: List of Arguments.

Modifier label	Meaning	Example
`nmod`	noun	'*John is a linguistics professor.*'
`amod(simple)`	simple adjective	'*John is an old man.*'
`amod(comparative)`	comparative adjective	'*John is an older man than Bill.*'
`amod(superlative)`	superlative adjective	'*John is the oldest man*'
`advmod`	adverb	'*John slept well.*'
`ppmod`	preposition	'*John slept in the park.*'
`whmod`	relative clause	'*The man who loves Mary slept.*'
`possession`	possession	'*John's car is old .*'

Table 6.3: List of Modifiers.

of the restrictor set (in case the specified argument is to be interpreted as a QNP (Section 5.2)). For instance, `*(some(N))` is specifying an indefinite set of entities whose cardinality is `N`, `*(the(N))` says the specified argument refers to a set of `N` entities, and `*(proRef(N))` indicates that the specified item is a referring pronoun to `N` entities. The cardinality is given as a numerical (in)equality, e.g. '*a man*' has cardinality `=1`, '*some men*' has cardinality `>1`. Examples of some other special specifiers that supply other kinds of information include: `*(zero)` which specifies that the argument is vacant, as in the case of a non-overt subject, and the time specifier: `*(time(Timeless,`




```
[word(.)/den(.),
   arg(claim,
       *(time(-,+,[simple,past],B)=1),
       B,
      [word(ate)/den(eat),
         arg(dobj,*(some(=1)=2.0),B,
            [word(peach>0 )/den(peach),
             modifier(amod(simple),*(-),[word(ripe>0)/den(ripe)]),
             modifier(C, D, [word(a)/den(a)])]),
         arg(subject,*(name(=1)=1000),B,
            [word(John)/den(John)])])]
```

Figure 6.15: *'John ate a ripe peach.'*

`Def,[Aspect| Tense_sequence], Time_var))`. The `Time_var` of a time specifier is again for sharing the time between an event and its arguments. The place holder `Def` and the list `[Aspect| Tense_sequance]` are (as have been illustrated in Section 5.5.3) for describing the temporal information: aspect, tense sequence, and definiteness sign (i.e. whether the described time is referential or existential). Finally, `Timeless` is for experimental reasons and will be discussed further in Chapter 10.

The second part of specifiers is an assignment of a `Scope_score`. Based on the discussion given in Section 5.9, the `Scope_score` is a value that indicates the scope of the argument being specified with respect to other scope-bearing items. Since the aim of scope scores is to indicate the order of precedence, their given specific values are not important, but their relative values are, with smaller scores specifying wider scope. See the DT of '*John ate a ripe peach.*' in Figure 6.15 for an example of all the above

Specifier Category	Specifier	Examples/notes
Universal	`*(forall)`	'***every*** *man loves cars.*''
	`*(bareplural(subject,_)`	'***men*** *love cars.*': bare plural in subject position = universal (Section 5.2)
Existential	`*(some(N))`	'***some man*** *slept in the park*', '***some men*** *slept in the park*'
	`*(num(N))`	'***three men*** *slept in the park*'
	`*(predet(least,N))`	'***at least three men*** *slept in the park*'
	`*(predet(most,N))`	'***at most three men*** *slept in the park*'
	`*(many)`	'***many men*** *slept in the park*'
	`*(of)`	'***two of the men*** *slept in the park*'
	`*(bareplural(dobj,_))`	'*men love* ***cars***.' bare plural in object position = existential (Section 5.2)
Default	`*(most)`	'***most men*** *love cars.*''
	`*(few)`	'***few men*** *love cars.*''
Referential/Definite	`*(the(N))`	'***the*** *man slept.*', '***the*** *men slept.*': **all** NPs are taken to denote sets, `N` here is the cardinality of the set.
	`*(proRef(N))`	'***he*** *loves* ***her***.'
	`*(name)`	'***John*** *loves* ***Mary***.'
	`*(both)`	= '*the two*'
	`*(either)`	= '*one of the two*'
	`*(neither)`	= '*not one of the two*'
Time	`*(time(Timeless, Def,[Aspect|Tense_sequence], Time_var))`	See Section 5.5.3
Adjectives	`*(adjSpec)`	needed for predicative uses of adjectives: '*John is happy*' (Section 5.6).
Negation	`*(not)`	'*John does* ***not*** *like peaches.*': this is not normally seen as a specifier – see Section 5.5.2
	`*(no)`	'***no*** *man is an island.*'
Miscellany	`*(zero)`	for zero items as non-overt subjects
	`*(of)`	for complex Dets such as '***six of the*** *men*'
	`*(whPron)`	'*who*', '*which*'

Table 6.4: List of Specifiers.

notions.

6.4 Summary

In this chapter, we have discussed the grammar used for generating our DTs and the reasons behind choosing this grammar over an off-the-shelf statistical parser. We have

also identified (CO2.1) a number of semantic features that ought to be encoded on DTs including some features that are widely acknowledged to be semantically relevant (e.g cardinalities of GQs, the shape of time information, and scope scores) but were missing from the parser's output, and others that have been identified in Chapter 5 (e.g. semantic class of adjectives, attitude verbs' entailment patterns and polarity marking) that we believe are crucial for constructing normalized DTs (Chapter 7). Finally, we have illustrated trees' general structure and explained in detail what each part of it means and what are the possible values for each of these parts.

Chapter 7

Tree Normalization

The previous chapter explained how we obtain our basic representations, which are DTs. As with any of the standard logics, it is convenient to convert the basic representation to a form that is well-suited to the demands of the theorem prover with which they are to be used, e.g. formulae of standard logics are typically converted to quantifier-free form or clausal form. Among the NLI systems discussed in Section 4.3 it is only Abzianidze's (2015; 2016; 2017a) work that incorporates theorem proving, and hence this is the only system that normalizes NLI problems into forms (LLFs) that are suitable to use within their inference engine (LangPro). While the expressiveness of LLFs goes beyond the polarized trees (trees marked with polarities) of MacCartney's (2009) NatLog system and Hu et al.'s (2019a) MonaLog system, they still say little about the semantic intricacies discussed in Chapter 5. However, LangPro (as its performance on the FraCaS dataset shows) can still handle deep inferences by having many rules (around 80) that are devised specifically for application to certain linguistic constructions.

What we do in this work is, to some extent, the opposite. Our basic MRs, as shown in the previous chapter, are DTs that are decorated with fine-grained semantic features including: time information, cardinalities of GQs, scope scores, semantic classes, entailment patterns, etc. What we do with these decorated trees is to apply a set of transformational stages that turn them into forms that the inference engine, which consists of only a handful of inference patterns, desires. Therefore, in this chapter, we explain the transformational stages that we have designed (CO2.2) to turn DTs into the prover's desired form; the *inference friendly forms* (IFFs). These stages are illustrated in Figure 7.4, and after a brief introduction (Section 7.1) each of them is explained separately.

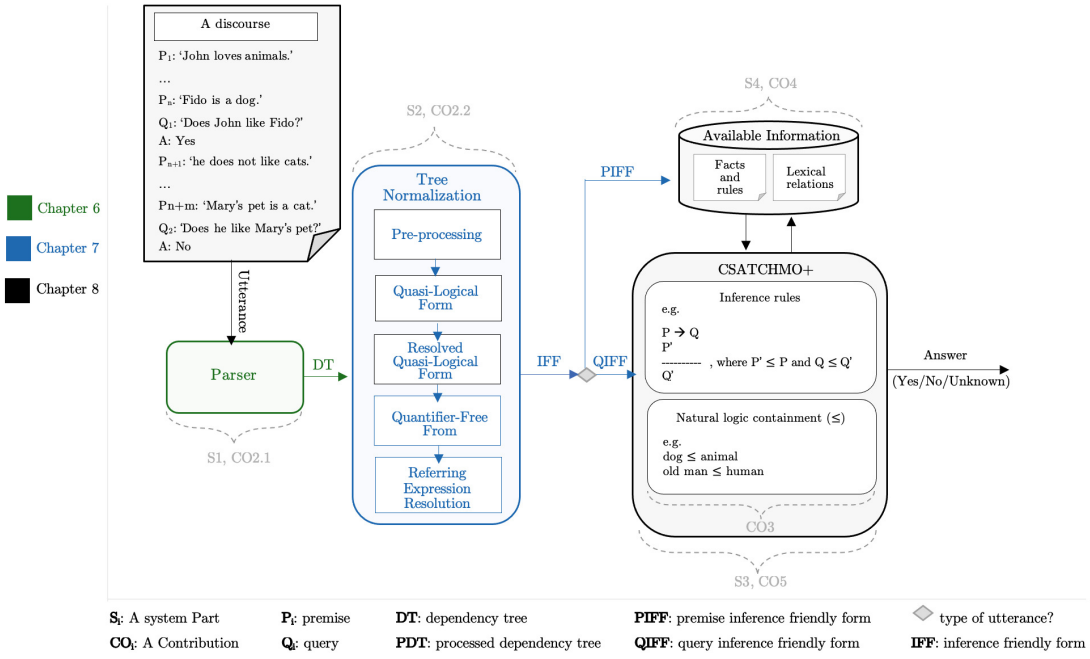


Figure 7.1: Dataflow through the inferential system.

7.1 Introduction

Most theorem provers require the application of a series of *normal form* rules which convert a (comparatively) readable formulae like in Figure 7.2 into flatter forms (e.g. quantifier-free form, clausal form, ...) as in Figure 7.3. The flatter forms make it easier for the inference engine to match facts and rules: in the original standard form, the fact that this rule will let you conclude that there was a loving event is buried deep in the formula, whereas in the flattened form it is immediately visible and can be used for indexing and matching.

```
exists(A,
       woman(A)
       & forall(B, man(B)
                   => exists(C, event(C, love)
                                & theta(C, object, A)
                                & theta(C, agent, B)
                                & aspect(now, simple, C))))
```

Figure 7.2: Ordinary logical formulae for '*every man loves a woman.*'

Similarly, our theorem prover, which is an adapted version of what is basically a FOL theorem prover (Section 3.3), requires its input to be in a form the allows its facts

```
woman(#1)
man(B) => event(#2(B), love)
man(B) => theta(#2(B), object, #1)
man(B) => theta(#2(B), agent, B)
man(B) => aspect(now, simple, #2(B))
```

Figure 7.3: Normalized formulae for '*every man loves a woman.*'

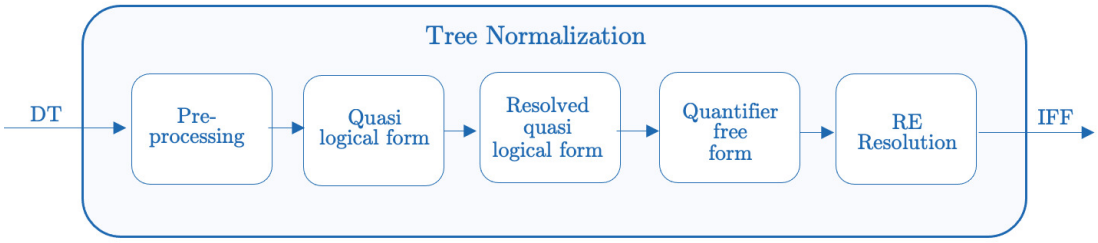
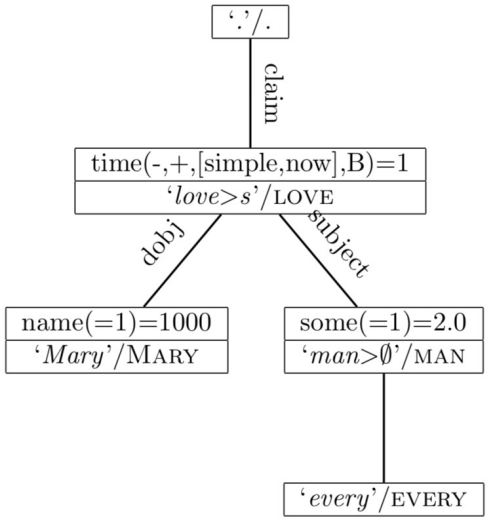


Figure 7.4: From a DT to an IFF.

and/or rules of inference to be easily matched and applied. Therefore, PDTs, such the one for '*every man loves Mary*' in Figure 7.5(a), go through a series of transformational stages (Figure 7.4) that convert them into flatter forms of *facts* and/or *rules* called IFFs, such as the ones in Figure 7.5(b). What these stages generally do (after a bit of pre-processing (Section 7.2)) is, first, group specifiers into definites, indefinites, and universals and turn them into scoped binding operators that bind the trees they specify (Section 7.3). Then, to ensure a preferred reading for the whole tree, a scope resolution is performed, making use of the scope scores assigned to operators (Section 7.4). Finally, tree parts marked by entirely either the definite or indefinite operators are fact-like and hence, their variables are turned into terms; definites by resolution (Section 7.7) while indefinites by *skolemization* (Section 7.5.3). Rule-like tree parts, which are marked by universal operators and a few other specific ones, keep their variables and include some kind of implicational symbol (Section 7.5.3).



(a)

```
** FACTS **
[fact({[female?1]@@A,[#,9]}, claimed),
fact({card,[#,9],=1}, assimilated),
fact({{name,[Mary?1]}@@A,[#,9]}, assimilated),
fact({{time,[now]},[#,8]}, assimilated)]
** FORWARDS RULES **
[({[man? -1]@@[#,8],B}
   => ({([love?1,{dobj,[#,9]},{subject,B}]@@{simple,[#,8]},[#,7,B])}: claimed))]
```

(b)

Figure 7.5: From a DT to an IFF example.

7.1.1 Notational Conventions

Some of the notations involved in the following discussions were explained in Section 5.1, Section 6.3 and here prior their use inside grey boxes. Table 7.1 list some of the frequently used ones and their use/meaning. Note that, in this chapter things that starts with a capital letter are either variables or place holders that could have different values in different instances. For example the general shape of specifiers is `*(Specifier_name=Scope_score)` and a possible instance is `*(forall=2)`.

Facts	**fact**{Restrictor, STATUS}	the general shape of facts and their parts.
Forward rule	A **=>** C:STATUS	the general shape of forward rules and their parts.
Backwards rule	C **<=** A:STATUS	the general shape of backward rules and their parts.
Time	[...]**@@T** or [...]**@@CTXT**	reads: at the the time given by T and at the time given by the context CTXT.
Polarity	[Word**?Polarity**]	are assigned at word level and they have 3 possible values: ?1 for positive polarity, ?-1 for negative and ?0 for no polarity.
Conjunction	{...} **&** {...}	
Skolem function	[#,SK]	A skolem function in which SK is a skolem constant.
Others	-->	we use this arrow in the definition of rules (given in grey boxes) to indicate the before and after forms.

Table 7.1: Some used notations.

7.2 Pre-processing

DTs are to be normalized into forms that are suitable for the inference engine to work with. Before that can be done, they have to undergo two forms of pre-preprocessing: (1) *removing* inessential items from the tree, and (2) *restructuring* some of its parts to facilitate follow up transformational steps.

7.2.1 Removing inessential terms

We start by removing several very obvious things. We remove the top-level punctuation mark, i.e. '.' or '*?*', since their task is to mark the whole utterance as being a claim or a query and this is encapsulated in the label they assign to the remainder of the tree. The words' inflected forms are also removed because the information they project about a word is already implicit in that word's denotation and specifier. For instance, in '*John ate a ripe peach*', Figure 6.15, word(peach>0) the marker indicates a singular peach and, at the same time, that word's specifier *(some(1)=2.0) says it is specifying one '*peach*' which is the same thing.

The other type of things to be deleted are dummy items. As explained in Section 6.2.2, some adjuncts function purely as modifiers, some purely as specifiers and some as both. Where an adjunct acts purely as a specifier, the information that it provides

is attached directly to the head node, and the node for the adjunct can be deleted. For example, in '*John ate a ripe peach*', Figure 6.15, the article '*a*' conveys the existence of some singular '*ripe peach*' in the specifier `*(some(1)=2.0)`, but has a dummy role (`C`) and hence it will be removed, and the tree will become as in Figure 7.6. Similarly, '*that*' in '*I know that she loves me.*' is syntactically a modifier, but it does not carry either a specifier or a modifier and hence is deleted at this point.

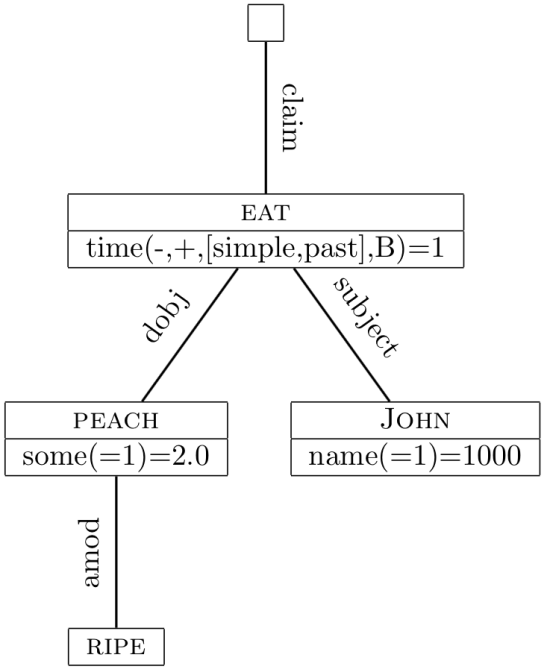


Figure 7.6: *'John ate a ripe peach.'* after removing excess information.

Lastly, once time specifiers have been built (as described in Section 5.5.3), from the tense markings, of the verb (again, that includes not only the inflectional affix but the auxiliary sequence as well), these markings become excess information and hence can be removed. Then the final result will be a verb and a time specifier. Take for example the sentence '*John has been working.*' and its DT in Figure 7.7. The sequence '*has been*' plus the affix '*-ing*' indicates a present perfect progressive. That tense is represented, according to our version of Reichenbach's system, as the list `[prog, past, now]` in which `prog` is the progressive aspect; `past` is the event time; and `now` is the reference time. That list, along with the definiteness sign `+` and the time variable `D`, all collectivity make up the specifier of the event WORK.

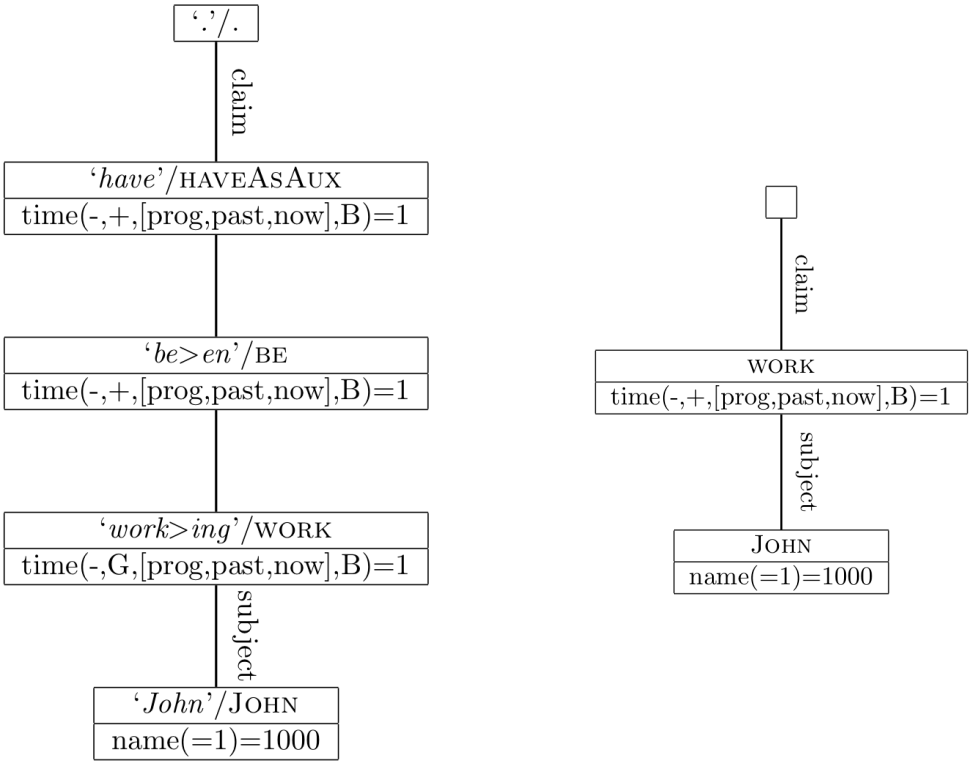


Figure 7.7: '*John has been working.*' before and after pre-processing.

7.2.2 Restructuring

This step is about a number of situations where two sentences with fairly different surface structures have very similar interpretations, e.g. '*The front door of my house is painted white*' and '*My house's front door is painted white*'. We deal with these by applying *tree-transformation rules* to ensure that the two versions of the utterance end up with the same trees.

Possession phrases, such as (7.1), are believed to mean exactly the same thing.

(7.1) a. John's friend slept.

b. The friend of John slept.

In order to obtain a unified treatment for both forms, we turn the possession marker '*s*', Figure 7.8(a) into its equivalent prepositional form as in Figure 7.8(b).

Copula sentences, as explained in Section 5.6, are to be turned into inferential equalities. Therefore, at this stage, the DTs of '*NP be NP*' a copula sentence is transformed into a tree where the copula is replaced with the equality sign (=) and the subject's

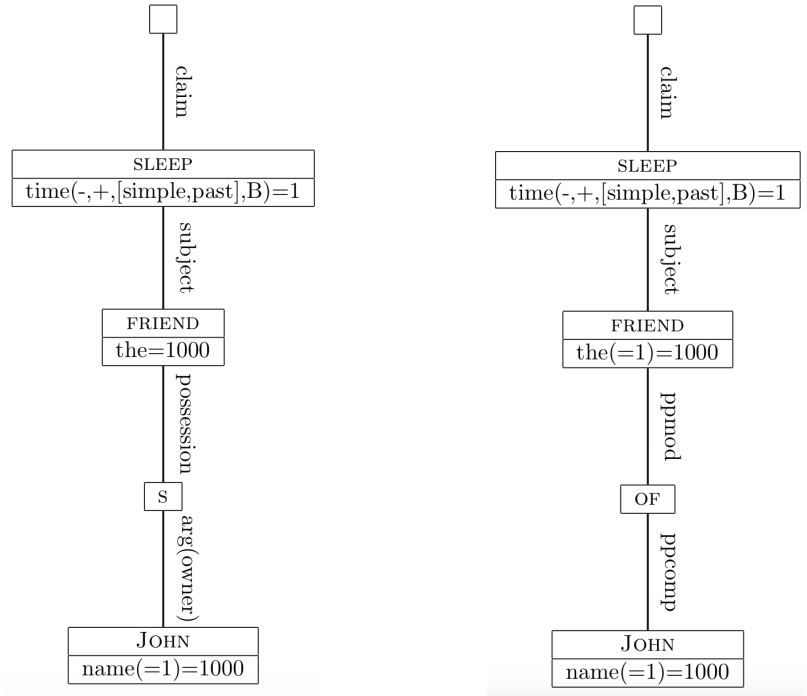


Figure 7.8: from '*John's friend slept.*' to '*The friend of John slept.*'

and the post copula's grammatical labels are replaced with the labels `eq2` and `eq1` respectively. For instance, compare the DT of '*John is a man*' (Figure 7.9(a)) and its reformed version in (Figure 7.9(b)).

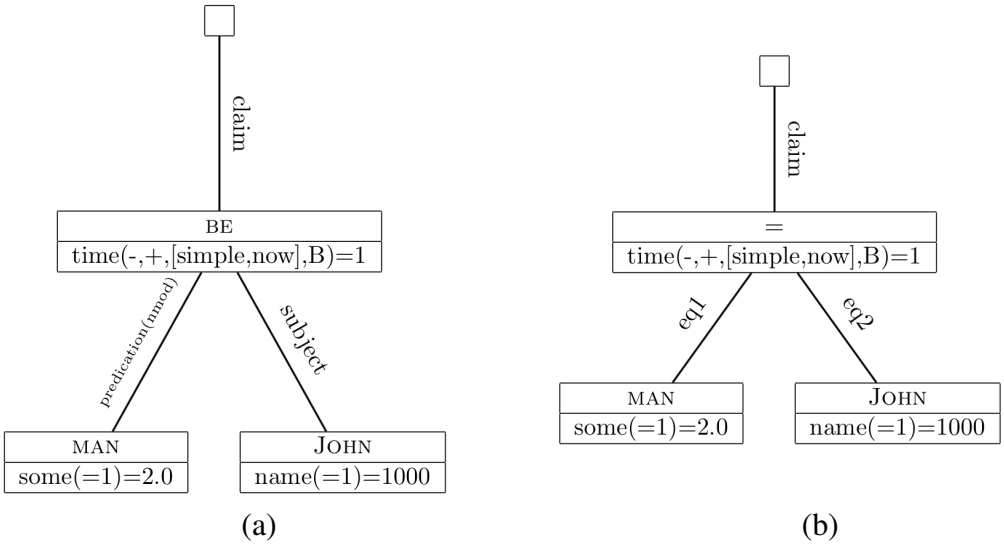


Figure 7.9: '*John is a man*' before and after restructuring.

Such reformation is not as straightforward for other forms of copula sentences

(‘*NP be AP*’ and ‘*NP be PP*’). That is, as previously discussed (Section 5.6), due to the fact that ‘*NP be AP*’ and ‘*NP be PP*’ copula sentences ought to undergo another kind of transformation; the transformation that makes both AP and PP modifiers for an indefinite hidden noun, before turning them into trees of equals. To demonstrate, consider the examples (7.2a) and (7.2b) and their DTs before and after reforming in Figure 7.10 and Figure 7.11 respectively. In the first, the post copula is an NP as its grammatical label (`pedication(`**`nmod`**`)`) shows. Hence, its transformation is pretty straightforward; relabelling.

(7.2) a. John is a man in the park.

b. John is in the park.

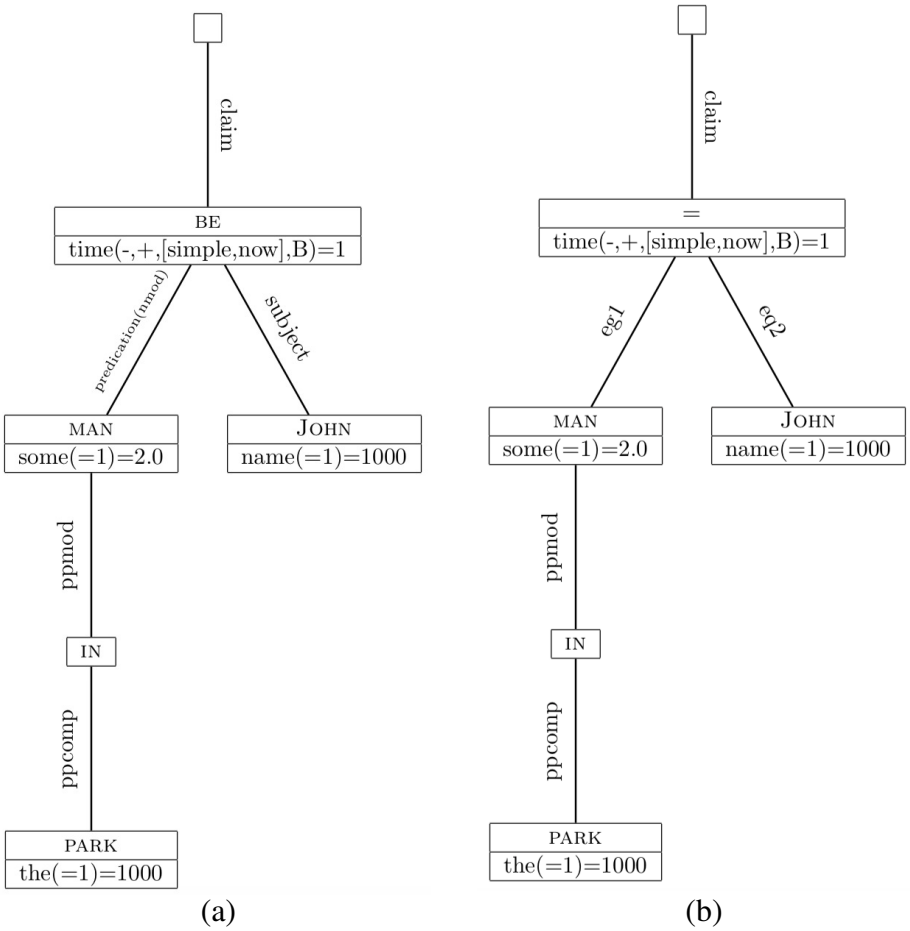


Figure 7.10: ‘*John is a man in the park*’ before an after restructuring.

On the other hand, the post copula of the second (7.2b) is a PP as its grammatical label indicates (`pedication(`**`ppmod`**`)`). Thus, the PP modifier is first transformed from

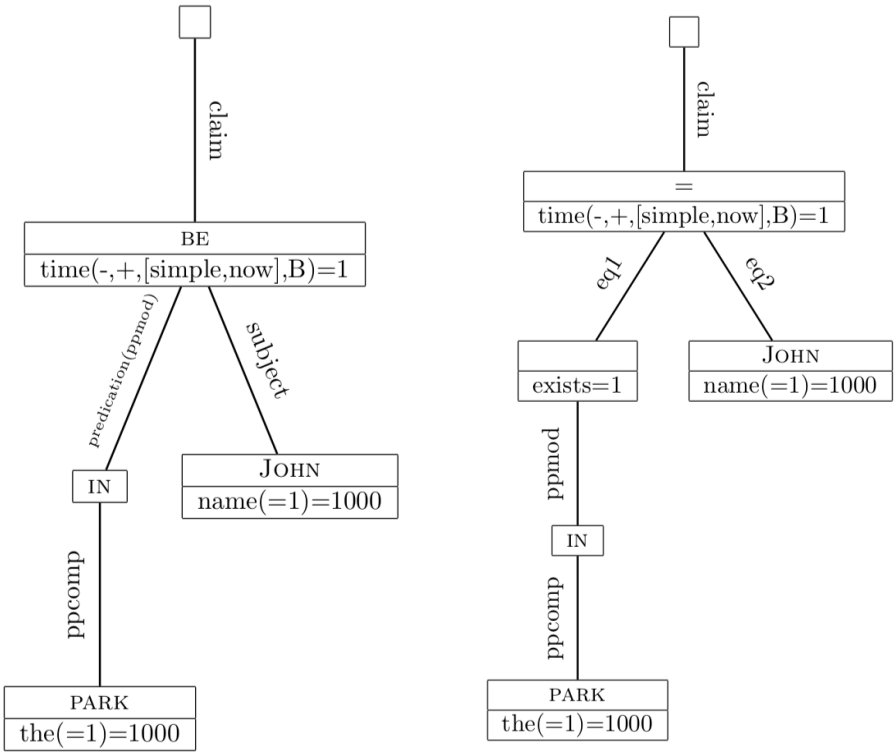


Figure 7.11: '*John is in the park*' before and after restructuring.

predicational into an attributive modifier for a hidden indefinite NP; that NP is the empty node with only a specifier (`exists`) in Figure 7.11 (b). After doing that, we can relabel the DT using the same machinery used for '*NP be NP*' sentences obtaining a structure that is similar to the one in Figure 7.10(b) where the only difference is that in the latter '*in the park*' modifies an actual NP ('*a man*'), while in the first it modifies a dummy hidden one ('*a thing*', '*an object*','*an individual*', etc.).

Finally, in view of the discussion in Section 5.7, ES are also to be turned into a special kind of copula sentences where the equality is between the expletive '*there*' and the post copula. For example, see the DT of (7.3a) and its transformation into a copula structure in Figure 7.12 (a) and (b) respectively. The restructured tree in 7.12 (b) expresses the existence of something (i.e. `X`) and the identity of that thing is '*a man in the park*'. Such restructuring has made the ES, in a sense, similar to its canonical subject-predicate counterpart in 7.3(b), where the DT of that counterpart also expresses the existence of '*a man in the park*' but in a more straightforward way.

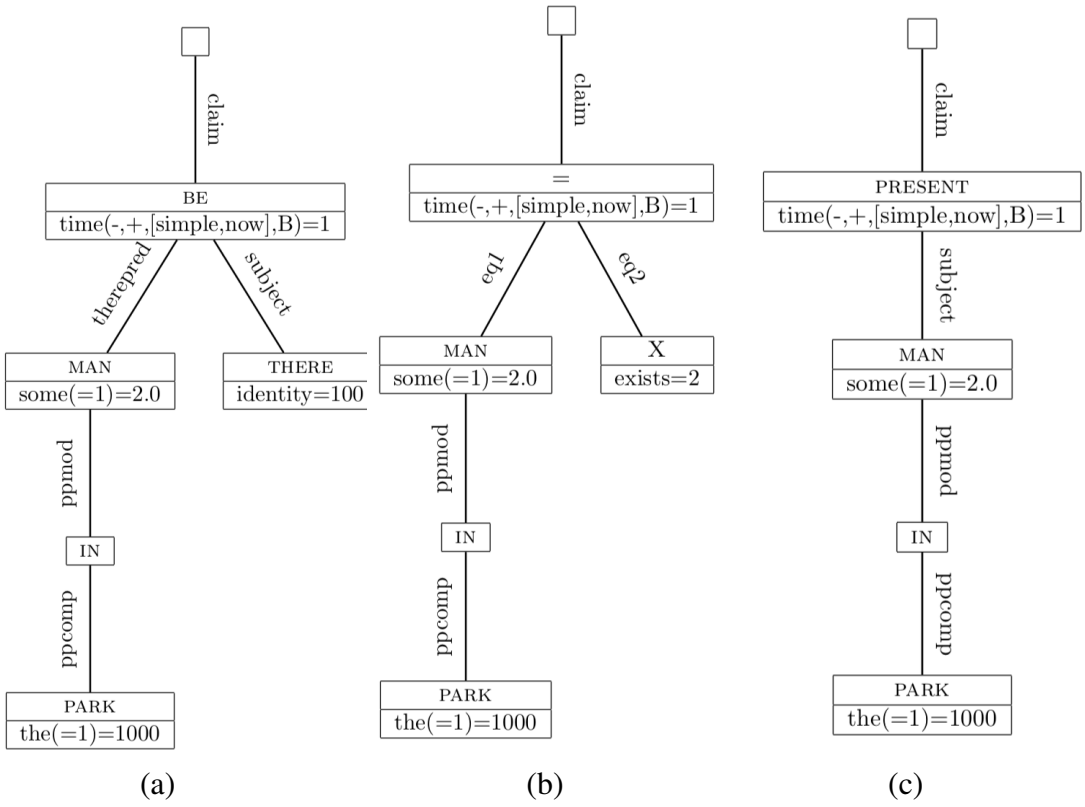


Figure 7.12: '*there is a man in the park.*' vs '*a man in the park (exists).*'

(7.3) a. '***There is*** *a man in the park.*'

b. '*A man in the park* ***(exists).***'

7.3 Quasi Logical Form (QLF)

As we have seen in the previous chapter, many aspects of the semantics of sub-trees are encapsulated in their specifiers; their definiteness, cardinality, scope, etc. We believe that the distinctions we need to make about a specified tree part (i.e. whether it is fact-like or rule-like, whether it needs to be introduced to the conversation or resolved to its referent, and what its order of precedence with respect to other tree-parts is) are all given in its specifier. Therefore, to keep such distinctions apparent, and to facilitate the final conversion into assert-able facts and rules, in this section specifiers are turned into in-situ quantifiers each called a `qq` (for `q`*uasi* `q`*uantifier*). That in turn converts the PDT containing them into an intermediate form that we called[1] *quasi logical form* (QLF).

[1]The choice of the form name, is motivated by the notational similarity to Alshawi and van Eijck [1989]; Alshawi [1992] quasi logical forms.

The general rule for turning specifiers into qqs is given in (QLF-R1). A qq consists of a wrapper (qq(...)) to mark its position within the tree and a scoped binding operator that binds its restrictor using a binding variable. The restrictor ST1 @@ Time_var is the specified sub-tree ST0 after being itself transformed into a QLF and attached to the shared Time_var. The binding operator assigned to a tree-part reflects to what group of specified trees that part belongs. Put differently:

- Tree-parts with definite specifiers (e.g.*(the...) and *(name...)), are facts-like that need resolving. Thus, their specifiers are turned into instances of the quantifier: qq(the...).
- Tree-parts with indefinite specifiers (e.g. *(num...) and *(some...)), are facts introducing, and hence, their specifiers are turned into instances of the quantifier: qq(exists...) .
- Tree-parts with universal specifiers (e.g. *(forall...)) and some other special ones (such as *(most...)), are rules introducing parts. Therefore, their quantifiers are instances of: qq(forall...) .

A simple example of these notations is given for the partial tree of '***every man*** *slept.*' in (7.4). Although the general rule for constructing qq forms is as given below, its formation most of the time involves some case specific transformation and tidying up operations and, hence they are grouped accordingly in the subsequent sections.

QLF-R1: qq general formation rule

qq form:

arg(Gram_label,*(Specifier_name=Scope_score),Time_var,ST0) **-->**
arg(Gram_label,**qq(**Operator=Scope_score , VAR :: {ST1 @@ Time_var, VAR}**)**)

scoped operator; binding var; restrictor

(7.4) Every man slept.

```
arg(subject, *(forall=2), B, [den(man)]) -->
arg(subject, qq(forall=2, E::{[den(man)]@@B,E}))
```

7.3.1 QNPs

As discussed in Section 5.2, QNPs are to be represented as type $\langle 1 \rangle$ quantifiers such that those with existential interpretations are instances of `exists` and will have the cardinalities of their denoting sets as part of their restrictor. Cardinalities, as illustrated in Section 6.3, are included in specifiers. Therefore, `qq forms` of QNPs are obtained by applying the rule in (QLF-R2). This rule is as (QLF-R1) except that the cardinality `N` given in the specifier, is turned into a *cardinality clause* that has the form {`card,VAR,N`} and is added to the `qq form` as a conjunct to the restrictor. For instance, the QNP in (7.5a) involves a set of two men (`*(the(=2)=1000)`) and hence, its cardinality clause will be {`card,S,=2`}, while in (7.5b) the QNP denotes a singleton set (`*(some(=1)=2.0)`), thus its clause {`card,S,=1`}. The description of the size of `N` can be something other than a simple equality expression. Consider (7.5c) for instance. The quantifier '*at least three*' has its cardinality expressed as `(>=3)` and hence the clause {`card,S,>=3`}. Moreover, some quantifiers are numerically inexpressible, such as '*a few men*' (in 7.5d). In such cases the cardinality clauses will include the determiner's name as the size description with appropriate background rules to interpret this cardinality when carrying out inferences.

QLF-R2: QNPs formation rule (set)

```
arg(Gram_label, *(Specifier_name(N)=Scope_score), Time_var, ST0) -->
arg(Gram_label,qq(exists=Scope_score, VAR ::
                              {ST1 @@ Time_var, VAR}&{card,VAR,N }))
```

(7.5) a. Two men slept.

```
arg(subject, *(num(=2)=2), B,[den(man)]) -->
arg(subject, qq(exists=2, S :: {[den(man)]@@B,S} & {card,S,=2}))
```

b. A man slept.

```
arg(subject, *(some(=1)=2.0), B,[den(man)) -->
arg(subject,qq(exists=2.0, S :: {[den(man)]@@B,S} & {card,S,=1}))
```

c. At least three men slept.

```
arg(subject, *(predet(least,3)=0.5), B,[den(man)])  -->
arg(subject,qq(exists=0.5, S :: {[den(man)]@@B,S} & {card,S,>=3}))
```

d. A few men slept.

```
arg(subject, *(num(few)=2), B, [den(man)]) -->
arg(subject, qq(exists=2, S :: {[den(man)]@@B,S} & {card,S,few}))
```

A QNP could also denote a set of entities that is a subset of another set. An example of such QNPs is expressions of the form `(subset` **`of`** `set)` as in (7.6). The QNP in (7.6) involves a set of one man, `D`, that is a subset of a larger known group of men, `C`. In such cases, the subset `qq form` is included inside the `qq form` of the main set according to the rule (QLF-R3). Moreover, it will have, in addition to the cardinality clause that shows its size, a *subset clause* of the format {`subset,`**`C`**`,D`} that links it to the main set.

QLF-R3: QNPs formation rule (subset)

```
arg(Gram_label, *(of=Scope_score1), Time_var,
        [den(of),
        arg(ppcomp,*(Specifier_name(N2)=Scope_score2,Time_var,ST2),
        modifier(numAsMod, Factivity, [den(N3)])])

-->
qq(Operator=Scope_score2, VAR2 ::{ST2@@Time_var, VAR2} & {card,VAR2,N2}),
  arg(Role, qq(exists=2, VAR3::{card, VAR3, N3}&{subset,VAR2,VAR3}))
```

(7.6) One of the men slept.

```
arg(subject,
    *(of=2),
    B,
    [den(of),
    arg(ppcomp, *(the(>1)=1000), B, [den(man)]),
    modifier(numAsMod, *(+), [den(=(1))])]) -->

qq(the=1000, (C :: ({[den(man)]@@B,C} &{card,C,>1} )),
   arg(subject, qq(exists=2, D::{card, D, =1} & {subset,C,D})))
```

Finally, for a number of quantifiers the cardinality information is omitted. That includes universals which do not assume existential import (e.g. (7.4)), default quantifiers (e.g. (7.8)) and mass nouns (e.g. (7.7)). For others, including the negation quantifier '*no*' (e.g. (7.9)) and BPs in object position (e.g. (7.10)) the cardinality information is left unspecified. In addition, both kinds of quantifiers keep their specifiers' names as the operator to ensure distinctive treatments in Section 7.5.

(7.7) John loves music.

```
arg(dobj, *(baresingular=2.1), B,[den(music)]) -->
arg(dobj, qq(baresingular=2.1, S :: {[den(music)]@@B,S}))
```

(7.8) Most men sleep.

```
arg(subject, *(most=2.0), B,[den(man)) -->
arg(subject, qq(most=2.0, S :: {[den(man)]@@B,S}))
```

(7.9) No man sleeps.

```
arg(subject, *(no=0.9), B,[den(man)]) -->
arg(subject, qq(no=0.9, S :: {[den(man)]@@B,S} & {card,S,X}))
```

(7.10) Mary loves cats.

```
arg(dobj, *(bareplural(dobj,B)=2.1), A, [den(cat)]) -->
arg(dobj, qq(bareplural(dobj,B)=2.1,D::{[den(cat)]@@A,D}&{card,D,X}))
```

7.3.2 DNPs

DNPs, as explained in Section 5.3, are *referring expressions* (REs) that need resolving (i.e. their referents need to be found). Therefore, as depicted in rule (QLF-R4), their `qq forms` have `the` as the binding operator. Definite descriptions, as illustrated in Section 5.3.1,have a representation that is the same as their indefinite counterparts except the use of a different operator (see (7.11) for example). Proper names, Section 5.3.2, are represented as definite descriptions that can be read as '*the thing named X*' as in (7.13). Finally, anaphoric pronouns (Section 5.3.3), commonly consider gender, cardinality, and recency factors for their resolution. While gender is expressed in the restrictor (as in (7.12)) and cardinality expressed by its clause, recency is expressed using an indicator clause {`recent,V`}.

QLF-R4: DNPs formation rule

```
arg(Gram_label, *(Specifier_name(N)=Scope_score), Time_var, ST0) -->

The NP:
arg(Gram_label,qq(the=Scope_score, VAR ::
                {ST1 @@ Time_var, VAR} &{card,VAR,N }))
Proper names:
arg(Gram_label,qq(the=Scope_score, VAR ::
                {{name,ST1} @@ Time_var, VAR} &{card,VAR,N }))
Anaphoric pronouns:
arg(Gram_label,qq(the=Scope_score, VAR ::
                {ST1 @@ Time_var, VAR} & {recent, VAR} &{card,VAR,N }))
```

(7.11) The man is sleeping.

```
arg(subject, *(the(=1)=1000), T, [den(man)]) -->
arg(subject, qq(the=1000, V :: {[den(man)]@@T,V} & {card,V,=1}))
```

(7.12) He slept.

```
arg(dobj, *(proRef(=1)=1000), T, [den(male)])  -->
arg(subject, qq(the=1000 , V ::
                    {[den(male)]@@T,V} & {recent,V}& {card,V,=1}))
```

(7.13) John is sleeping.

```
arg(subject, *(name(1)=1000), T, [den(John)])  -->
arg(subject, qq(the=1000, V :: {{name,[den(John)]}@@T,V}&{card,V,=1}))
```

Other determiners that combines with NPs creating definiteness and are expressible in terms of the article '*the*' are '*both*' and '*either*'. The determiner '*both*' will become '*the two..*' and '*either*' is the same as '*one of the..*' and hence their `qq` forms are as exemplified in (7.14) and (7.15).

(7.14) Both men are sleeping.

```
arg(subject, *(both=-10), T, [den(man)]) -->
arg(subject, qq(the=-10, S :: {[den(man)]@@T,S} & {card,S,=2}))
```

(7.15) Either man slept.

```
arg(subject, *(either=-10), T, [den(man)]) -->
qq(the=-10, C :: ({[den(man)]@@T,C} &{card,C,>1} ),
  arg(subject, qq(exists=2, D::{card, D, =1} & {subset,C,D}))
```

7.3.3 Events and Time

Following the choice of Davidson's style events (given in Section 5.5) and the discussion of turning a tensed event into a simple event descriptor and a time specifier in Section 6.2.2, tensed events are represented using two nested `qq forms`. The outer `qq` is for time, while the inner one is for the event itself. The formation rule for these nested `qq`s is given in rule (QLF-R5) and exemplified in Figure 7.13, such that:

- The `Def` sign in the time specifier, determines whether the time given by the specifiers is referential or existential, and hence indicates what operator is to be used for binding the time restrictor. In other words, the operator is `the` if `Def` value is `+`, otherwise it is `exists`.
- The time restrictor has the form {{`time,[Tense_seq]`}, `T`}.
- Following the time restrictor comes the event's `qq` in which the bound event restrictor is associated with the time at which it occurred using the *context clause*: `@@{ASPECT, T}`.

Figure 7.13 shows the QLF of the sentence '*John buttered the toast*' where both `qq` forms of quantifiers and time specifiers are integrated.

QLF-R5: Events formation rule

```
...,
    *(time(Timeless,Def,[Aspect|Tense_seq],T)=Scope_score),
        T,
          ST0) -->
(a)Referential Time (Def= +):
...,
qq(the =Scope_score, T :: {{time,[Tense_seq]}, T},
       qq(exists= 5, E, {ST1 @@ {ASPECT, T}, E}))

(b)Existential Time (Def= - or Variable):
...,
qq(exists =Scope_score, T :: {{{time,[Tense_seq]}, T},
              qq(exists= 5, E, {ST1 @@ {ASPECT, T}, E}))
```

```
...,
   *(time(-,+,[simple,past],A)=1),
   A,
   ([den(butter),
     arg(dobj, *(the(=1)=1000), A,[den(toast)]),
     arg(subject, *(name(=1)=1000), A, [den(John)])])  -->

...,
qq(the=1, A::{{time,[past]},A},
  qq(exists=5, B,
    {(([den(butter),
     arg(dobj,qq(the=1000,(C:: ({[den(toast)]@@A,C}& {card,C,=1})))),
     arg(subject,qq(the=1000,D::{{name,[den(John)]}@@A,D}& {card,D,=1}))]
    @@ {simple,A}),B)}))
```

Figure 7.13: QLF of '*John buttered the toast.*'

Copula Events

As established previously, in many places copula sentences are to be turned into equalities. Therefore, as in Figure 7.14, the equality `=` and its arguments `eq1` and `eq2` (which are placed on either side of the `(=)` and have been turned into `qq forms` themselves) will be existentially quantified inside the `qq` form of the time as any other event.

Negated Events

Negated events, such as in Figure 7.15(a), appear on the tree as arguments of the word '*no*' of type `negComp`. Their `qq` forms, as in Figure 7.15, are simply constructed by nesting the event's `qq form` inside the `qq(not=0.9, ...)` wrapper, and omitting the

```
...,
   *((time(-, +, [simple, now],A)= 1)),
      (arg(eq1, *(some(=1)=2.0), A, [den(man)])
      =
      arg(eq2, *(name(=1)=1000), A, [den(John)]) -->

qq(the=1,A::{{time,[now]},A},
   qq(exists=5,B, {(((
     arg(eq1,qq(exists=2.0, C::{[den(man)]@@A,C} & {card, C,=1}))
     =
     arg(eq2, qq(the=1000,D::{{name,[den(John)]}@@A,D} & {card,D,=1})))
   @@ {simple,A}),B)}))
```

Figure 7.14: QLF of '*John is a man*'.

negation word and the `negComp` argument wrapper as they are no longer needed. As with the quantifier '*no*', the negation word '*not*' introduces a negation rule (see Section 5.5.2). '*not*' does not bind any restrictor, but it does have a scope operator `not=0.9` in the form to determine what is in the scope of negation and hence ought to be part of the rule and what is not.

```
arg(claim,
    *(not=0.9),
    A,
    [den(not),
     arg(negComp,
          *(time(-,+,[prog,now],A)=1),
          A,
          [den(sleep), arg(subject, *(the(=1)=1000), A, [den(man)])])])  -->

qq(claim=0,
 qq(not=0.9,
   qq(the=1, A::{{time,[now]}A},
     qq(exists=5, B,
       {[den(sleep),
           arg(subject, qq(the=1000, C:: {[den(man)]@@A,C}& {card,C,=1}))]
         @@{prog,A} ,B}))
```

Figure 7.15: QLF of '*the man is not sleeping.*'

7.3.4 Utterance

Utterance labels, although they are only type indicators, do also have `qq forms` given in (QLF-R6) rule. Similar to '*not*', their form is basically a scoped operator wrapper: `qq(Utt_type= 0,...)`. The scope blocks any quantifiers other than `the` from escaping the scope of the utterance marker (Section 7.5). As for what the utterance operator wraps, it is `ST1` which is the QLF form of the sub-tree `ST0` with respect to the time `Time_specifier`. Figure 7.16 shows an example of an utterance's QLF.

QLF-R6: Utterances formation rule

```
arg(Utter Type, Time_specifier, T, ST0) -->
 qq(Utter Type= 0, ST1),
```

```
arg(claim,
     *(time(-,+,[prog,now],A)=1),
     A,
     [den(sleep), arg(subject, *(the(=1)=1000), A, [den(man)])]) -->
qq(claim=0,
   qq(the=1, A::{{time,[now]},A},
      qq(exists=5, B,
         {[den(sleep),
              arg(subject,qq(the=1000,C :: {[den(man)]@@A,C}& {card,C,=1}))]
            @@ {prog,A},B})
```

Figure 7.16: QLF of '*the man is sleeping.*'

7.3.5 SubCs

QLFs are to be resolved in the next step, Section 7.4, and that involves collecting all of their `qq`s, and then applying them in a certain order to the main proposition. However, as explained in Section 5.8, in this thesis we deal not only with simple sentences, but also with ones with SubCs. These SubCs are mostly sentential and could have their own QNPs. Moreover, as seen in Section 5.9, not all QNPs are allowed to escape the scope of their containing clause. Therefore, beside the `qq` formation rules given above, there is a *scope control* rule that ensures that SubCs' `qq`s are resolved locally before resolving the whole utterance. The rule is given in (QLF-R7). It wraps an `opaque/raw` operator around the QLF of the intended SubC and provides an `Escape_score`. That score acts as a condition under which a `qq` can escape its containing clause and get

resolved with respect to the whole utterance's qqs. opaque and raw do the same job towards their embedded QLFs when it comes to scope controlling and resolving. However, after scope resolution opaque forms are to be further transformed into quantifier-free forms then IFFs as any other resolved QLF, but those with raw wrapped around them will stay in their resolved QLF until used in proofs (see Section 8.5 for further discussion)

(QLF-R7): Opaque general formation rule

```
opaque(SubC QLF,Escape_score) Or
raw(SubC QLF,Escape_score)
```

The considered types of SubCs, which are RC and attitude verbs' complements (Section 5.8), are all sentential i.e. involve events. Thus, their QLFs' are generally obtained by applying (QLF-R5) rule, but with some type specific considerations:

- **RCs**, as shown in Figure 7.17, are intersective modifiers to nominals. Thus, their opaque forms are conjoined to the nouns they modify, see (QLF-R7.1). Moreover, RCs are tensed and have the relative pronoun as their subject/object. Therefore, when turned into a QLF they have their own time. However, the relative pronoun, as the example in Figure 7.17, is to be resolved to the intended referent when that referent is resolved itself (see Section 7.7).

- An **implicative clause**, as shown in Figure 7.18, is a tenseless complement of an implicative verb (see (QLF-R7.2)), and has a zero (non-overt) subject. Thus, when turned into a QLF, the clause inherits the time of the implicative verb and its zero-subject gets resolved to the subject of the implicative in Section 7.7.

- The complements of **propositional attitudes** (the ones discussed in Section 5.8.2) are tensed and have overt subjects, thus their QLFs are results of direct applications to rule (QLF-R5).

(QLF-R7.1): RCs opaque formation rule

```
...[NP , modifier(whmod, *(+), arg(whclause, Time_sepcifier,_,ST0))]...
                                                 SubC
-->
...NP & opaque(QLF of SubC, Escape_score) ...
```

```
arg(claim,
    *(time(-,+,[simple,past],B)=1),
    B,
   [den(sleep),
    arg(subject,*(the(=1)=1000), B,
    [den(man),
    modifier(whmod, *(+),
          arg(whclause,
           *(time(-,+,[simple,now],D)=1),
           D,
           [den(love),
           arg(dobj, *(name(=1)=1000), D, [den(Mary)]),
           arg(subject, *(whPron=0,9), D, [den(who)])])])])]
                        Subject relative pronoun
-->
qq(claim=0,
  qq(the=1,B::{time,[past],B},
    qq(exists=5,E,
    {[den(sleep),
      arg(subject, qq(the=1000, F:: {[den(man)]
        & opaque(
          qq(the=1,D::{time,[now],D},
           qq(exists=5, G, {[den(love),
            arg(dobj, qq(the=1000,H:: {{name,[den(Mary)]}@@ D,H})),
            arg(subject,qq(whPron=0.9, I:: {[den(who)]@@ D,I}))]@@ D,G}
              & {simple,D},0.5)
      @@ B,F}&{card, F,=1}))]@@{simple,B})))
```

Figure 7.17: QLF of '*the man who loves Mary slept.*'

(QLF-R7.2): Attitudes opaque formation rule

```
..., arg(xcomp(P),  Time_specifier, Time_var, ST0 )...
                              SubC
-->
...,arg(xcomp(P),opaque(QLF of SubC, Escape_score))...
```

```
arg(claim,
    *(time(-,+,[simple,past],A)=1),
    A,
    ([den(manage),
    arg(xcomp([1/1, -1/-1]),
        *(time(-,identity,[simple,X],A)=1),
        A,
        ([den(sleep),arg(subject, *(zero=0), A,[zero1])]),
                                  Zero subject
        arg(subject, *(name(=1)=1000), A, [den(John)])])))

-->
qq(claim=0,
   qq(the=1, A::{{time,[past]},A},
    qq(exists=5, B,
     {(([den(manage),
      arg(xcomp([1/1, -1/-1]),
       opaque(qq(exists=5, C
        {(([den(sleep), arg(subject, zero)]@@ {simple,A}),C)}), 0.5),
      arg(subject, qq(the=1000, D :: {{name,[den(John)]}@@A,D}&{card,D,=1}))]
    @@ {simple,A}),B)})))
```

Figure 7.18: QLF of '*John managed to sleep.*'

7.4 Resolved QLF (RQLF)

In the previous stage, PDTs have been turned into QLFs in which each `operator` was attached a `Scope_score`. In the current stage, these scopes are to be resolved. The scope resolution method, explained in Section 5.9, ensures that the `qqs` of a tree's QLF are collected and applied to the main proposition in such a way their scope scores are obeyed. To demonstrate, consider the QLF of '*John loves Mary*' and its RQLF in Figure 7.19. The process of obtaining the resolved form starts by scanning the QLF and collecting all the `qq` forms and placing them in a *qq stack*. This stack contains the collected `qqs` as two place terms: **`qq(`**`F_score,{Operator, VAR :: Restrictor}`**`)`**, in which `F_score` is just the floating point conversion of `Scope_score` for sorting purposes. However, as discussed in Section 7.3, not all `qqs` have the binding variable and/or restrictor, thus the minimal entry of the stack will be the scoped operator, `qq(F_score, Operator)`, as in the case of utterance labels. While collecting the `qqs` of an utterance, when an `opaque` form is encountered, the collecting process will get interrupted so that the `opaque`(ed) QLF gets resolved first. Once the embedded QLF is resolved, it is placed in the qq stack and then the collection process is resumed until the end of the main utterances's QLF. Moreover, among the collected `qqs`, the ones of

an event's arguments leave their binding variable in place, creating an abstracted main proposition to which all the collected quantifiers are applied.

The second step after building up the stack is to sort it in ascending order, as the smaller the score the wider the scope, and then to discard the scope scores as they are no longer needed. After that, the sorted stack is applied to the main proposition starting from the bottom of the stack; i.e. from the narrowest scope quantifier, creating the final RQLF.

```
QLF
qq(claim=0,
  qq(the=1, A::{{time,[now]},A},
    qq(exists=5, B, {[den(love),
        arg(dobj, qq(the=1000,C::{{name,[den(Mary)]}@@A,C}&{card,C,=1})),
        arg(subject,qq(the=1000,D::{{name,[den(John)]}@@A,D}&{card,D,=1}))]
    @@ {simple,A}, B})))

Ordered qq stack
[qq(0.0, claim),
 qq(1.0, {A::{time,[now],A}},
 qq(5.0, {exists,B}),
 qq(1000.0, {the,C::{{name,[den(Mary)]}@@A,C}&{card,C,=1}}),
 qq(1000.0, {the,D::{{name, [den(John)]}@@A,D}&{card,D,=1}}]

main proposition
 {(([den(love), arg(dobj, C), arg(subject, D)]@@ {simple,A}),B)}

RQLF
qq(claim,
  qq(the ,A::{{time,[now]},A},
    qq(exists, B,
       qq(the,C::{{name,[den(Mary)]}@@A,C}&{card,C,=1}),
       qq(the,D::{{name,[den(John)]}@@A,D}&{card,D,=1}),
      {(([den(love), arg(dobj, C), arg(subject, D)]@@ {simple,A}),B)})))
```

Figure 7.19: RQLF for '*John loves Mary.*'

7.5 Quantifier Free Form (QFF)

So far, we have illustrated how a PDT is turned into a form (RQLF) in which qqs are nested and sorted (see Figure 7.20 for example). As stated earlier, one goal behind the qq forms construction is to distinguish which parts of a PDT contribute to facts and which others contribute to rules. Now that the required RQLF has been obtained, this phase generally involves:

- Polarity marking.

- Removing all quantificational operators from the RQLF and turning the binding variables of indefinite ones into terms.
- Introducing logical symbols for *implication* `(=>)`, conjunction `(&)` and disjunction `(or)`, in place of the removed operators.
- Handling all special quantifiers that have been left till now such as '*most*', '*few*' and '*at most N*'.
- Discarding labels that are no longer significant, e.g. the `qq` labels in the RQLF, the `den()` and `arg()` wrappers so that every `den(Word)` will be turned into just `Word` and every `arg(Role, VAR)` into {`Role,VAR`}.

Figure 7.20 gives an example of the transition from RQLF to a QFF for the utterance '*John is sleeping*', and the subsequent sections demonstrate the actual steps for turning each part of a RQLF into a QFF.

```
PDT
arg(claim,
    *(time(-,+,[prog, now],B)=1),
     B,
    [den(sleep),
     arg(subject, *(name(=1)=1000),B, [den(John))])
RQLF
qq(claim,
  qq(the,B::{time,[now]],B},
    qq(exists,C,
      qq(the, (E::{{name,[den(John)]}@@B,E}&{card,E,=1}),
       ({[den(sleep),arg(subject,E)]@@{prog,B},C})))))
QFF
claim(the(A::{{time,[now]},A},
  the((B:: ({{name,[den(John?1)]}@@A,B}& {card,B,=1})),
    {([sleep?1,{subject,B}]@@{prog,A},[#, 3])})))
```

Figure 7.20: An Illustration to the DTs structural change from pre-processing till QFF.

7.5.1 Polarity Marking

As shown in Figure 7.1, one part of the inference engine is a natural logic-base matcher. The matcher supports the engine's proof process by making partial inferential decisions (e.g. '*dog*'$\models$'*animal*' and '*not an animal*'$\models$'*not a dog*') whenever necessary. To allow the matcher to do so, normalized trees should be marked with polarity. Different works in the literature have different ways of doing the marking (see Section 4.3). We chose

to mark RQLFs instead of PDTs because in the first QNPs are believed to have the right scope. Therefore, doing polarity composition will not require, for example, defining pattern as NatLog. Our process for polarity marking goes as follow:

- Polarities are marked at the word level; thus a (`den(Word)`) in a RQLF will become (`Word?Pol`) in the QFF, were `Pol` is either `1` for positive, `-1` for negative or `0` for non.
- The marking process starts with an initial value for polarity, which is `1` for claims and `-1` for queries (see Section 7.5.7).
- The RQLF of an utterance is scanned from the widest scope `qq` form to the lowest one.
- Each `qq` form will pass the default polarity to its restrictor and the remaining of the RQLF.
- If the `qq` form operator happens to be one of the polarities affecting words (e.g. `forall`, `most`, `few`, `no/not`), then it will perform some polarity compositions before passing polarities down to its restrictor and the rest of the RQLF. For example, `forall` in `qq(forall, R, P)` will reverse the received polarity before passing it to its restrictor `R` and pass it as it is to `P`. Such particular behaviour of `forall` is based on its monotonicity properties (given in Appendix A) whic is (↓↑).

Consider, for example, the RQLF of '*not every man sleeps*' in Figure 7.21. The marking process for this claim starts from the top with `1` as the initial polarity. When this polarity reaches `qq(not,...)`, it gets reversed into `-1` before passing it to the remaining of the RQLF. Then, when `qq(forall,..., ...)` gets that polarity (`-1`), it will reverse it back to `1` and use it to mark its restrictor, but pass it as it is for the remaining of the form (which is the `sleep` event). Therefore, the QFF of '*not every man sleeps*' has `man` marked with positive polarity, while `sleep` is marked with negative polarity.

```
RQLF
qq(claim,
  qq(not ↓,
    qq(def(+),A::{{time,[now]},A},
      qq(forall ↓↑,[#,8]::{[den(man)]@@A,[#,8]},
        qq(exists,B,{(([den(sleep), arg(subject, [#, 8])]@@ {simple,A}),B)}))))))
QFF
claim(
  (the(A::{{time,[now]},A},
    ({[man?1]@@A,[#,8]}
      => {(([sleep?-1, {subject,[#,8]}]@@ {simple,A}),B)}))
   => absurd))
```

Figure 7.21: Polarity marking example.

7.5.2 Referential operators i.e. `the`

Definite restrictors (bound with `the` operator), are meant to be resolved in Section 7.7. However, they will still appear in the QFF in their RQLF status except without the `qq` wrapper, see (QFF-R2). For example, check the representation of '*John*' in Figure 7.20.

QFF-R2: QFF of referential operators
Replace `qq(the, R)` by `(the, R)`, in which `R` will remain in its RQLF state.

7.5.3 `forall` and `exists`

FOL automated theorem provers usually require FOL formulas to be turned into *skolem normal form*s in which the existential quantifiers are removed and their bound variables are replaced with a *skolem function (SF)*[2]. Similarly, a `qq` form of an `exists` operator is removed and its variable is turned into a SF that has the form `[# | SK]`[3], leaving behind a skolemized restrictor that has that SF in place of the variable. In its simplest cases, when an `exists` is not in the scope of a `forall`, `SK` is a new constant. For instance, in the QFF of '*a man slept.*' (Figure 7.22), the SF for `[man]` and its cardinality is `[#,8]` and the event `sleep` is `[#,9]`. Note that when a variable is skolemized all of its occurrences are replaced by its SF. This is why the `subject`, in the same example has the function `[#,8]`.

[2] See [Fitting, 1990, p 187] for further illustration.

[3] A Prolog list can be represented either as a traditional listing of elements; [E_0,E_1,..E_n], or as `[H|T]` in which `H` is the head element of the list and `T` is the tail element(s).

QFF-R3: QFF of universal and existential operators

(i) Skolemise away existential operators `(exists)` and introduce the conjunctive and `(&)` after its restrictor.

(ii) Remove universal operators `(forall)` and insert the implication `=>` after its restrictor.

On the other hand, according to (QFF-R3)(ii), the removal of a `forall` introduces the rule symbol=> after its restrictor. Put differently, the removal of a universal operator introduces a rule whose antecedent is that operator's restrictor. In contrast to existential operators, the binding variable of a rule's antecedent will remain unchanged. For instance, the QFF for '*every man*' in '*every man sleeps.*' (Figure 7.23) shows that the binding variable `B` for '*man*' was left un-skolemized.

On the relation between `exists` and `forall` operators, if the first appears in the scope of the second, the `SK` of the first is no longer just a constant; it is the constant followed by a list of variable(s). These variables are the binding variables of the universal operators(s) that the `exists` falls under their scope. Thus, in the same example, '*every man sleeps.*' (Figure 7.23), the existentially quantified sleeping event is in the scope of `forall`, and hence its SF is `[#, 3, B]`.

```
RQLF
 qq(claim,
    qq(the,A::{{time,[past]},A},
      qq(exists,C::{[den(man)]@@A,C}&{card, C, =1},
        qq(exists, B, {([den(sleep),arg(subject,C)]@@{simple,A}, B)}))))
QFF
claim(the(A::{{time,[past]},A},
  (({[man?1]@@A,[#,8]} & {card,[#,8],=1})
    & {(([sleep?1, {subject,[#,8]}]@@ {simple,A}),[#, 9])})))
```

Figure 7.22: QFF for '*a man slept.*'

```
RQLF
qq(claim,
  qq(the,A::{{time,[now]},A},
    qq(forall,B::{[den(man)]@@A,B},
      qq(exists, C,{([den(sleep),arg(subject,B)]@@{simple,A},C)})))
QFF
claim(the(A::{{time,[now]},A},
  ({[man? -1]@@A,B}
    => {([sleep?1,{subject,B}]@@{simple,A},[#, 3, B])})))
```

Figure 7.23: QFF for '*every man sleeps*'.

7.5.4 Negation

Negated events, as discussed in Section 5.5.2, are to be dealt with constructively. Therefore, following standard practice in constructive logic, they are expressed as rules as indicated in (QFF-R4). For example, consider the RQLF of '*some man did not sleep.*' in Figure 7.24: the negated event which is nested inside the `qq(not,...)` wrapper becomes the antecedent of a rule to *absurd*. With that being said, the antecedent of that rule will be treated as the antecedent of any other rule, and hence all its binding variables will remain un-skolemized. The same applies for sentences with the negation quantifier '*no*', such as in '*no man slept*' (Figure 7.25). However, this time '*no*' has a restrictor. That is, in addition to being part of the negation rule's antecedent, the negated restrictor (the set of men: `{[den(man)]@@B,A}` in the example) has a variable as its size, `{card, A, D}`, resulting in a rule that can be read as follows: *for a set of men `A` of any size `D`, it is absurd for a sleeping event that involves that set of men at that time to exist*.

QFF-R4: QFF of negation operators
(i) Replace `qq(not, P)` by `(P'→ absurd)` (ii) Replace `qq(no, R, P)` by `((R' & P')→absurd)` Where `R'` and `P'` are the QFFs for `R` and `P` respectively:

```
RQLF
qq(claim,
  qq(exists, (B::{[den(man)]@@A,B}&{card,B,=1},
    qq(not,
      qq(the, A::{{time,[past]},A},
        qq(exists,C,{[den(sleep),arg(subject,B)]@@{simple,A},C}))))))
QFF
claim(
  (({[man?1]@@A,[#,8]}& {card,[#,8],=1})
    &(the(A::{{time,[past]},A},
      {(([sleep?-1, {subject,[#,8]}]@@ {simple,A}),C)})=> absurd)))
```

Figure 7.24: QFF for '*some man did not sleep.*'

```
RQLF
qq(claim,
  qq(no, (A::{[den(man)]@@B,A}&{card, A, D}),
    qq(the, B::{{time,[past]},B},
      qq(exists,C,{[den(sleep),arg(subject,A)]@@{simple,B},C}))))
QFF
claim(
  ({[man? -1]@@B,A}&{card,A,D}
    & the(B::{{time,[past]},B},
      {(([sleep?-1, {subject,A}]@@ {simple,B}),C)}))=> absurd)
```

Figure 7.25: QFF for '*no man slept.*'

7.5.5 Defaults

In light of the discussion in Section 5.2.2, the quantificational operators `most` and `few` are both turned into Reiter's style defaults as indicated in (QFF-R5). Therefore, considering the example '*most birds fly*', this step will break down the default utterance into three parts: (1) prerequisite, (2) consistency conditions, and (3) consequent. Thus, the QFF of that example, in Figure 7.26, will be read as follows: *at the time given by* `A`*, if* `B` *is a bird and it is consistent to assume that* `B` *flies, then we can infer that* `B` *flies.* The same goes for the example '*few men sleep*' in (Figure 7.27) that, as explained in Section 5.2.2, can be interpreted as '*most men do not sleep*'. Thus, the default rule in Figure 7.27 can be read as: *at the time given by* `A`*, if* `B` *is a man and it is consistent to assume that* `B` *does not sleep, then we can infer that* `B` *does not sleep.*

QFF-R5: QFF of '*most*' and '*few*'

(i) Replace `qq(most, P, Q)` by `((P'& consistant(Q'))=> Q') .`

(ii) Replace `qq(few, P, Q)` with `qq(most, P, not(Q))` then do (i).

Where `P'` and `Q'` are the QFFs for `P` and `Q` respectively.

```
RQLF
qq(claim,
  qq(the,A::{{time,[now]},A},
    qq(most,B::{[den(bird)]@@A,B},
      qq(exists, C,{([den(fly),arg(subject,B)]@@{simple,A},C)})))
QFF
claim(
  the(A::{{time,[now]},A},
    ((({[bird?0]@@A,B}
        prerequisite
      & consistent({(([fly?1, {subject,B}]@@ {simple,A}),[#, 3])}))
                          consistency condition
        => {([fly?1,{subject,B}]@@{simple,A},[#, 3])}))
                          consequent
```

Figure 7.26: QFF for '*most birds fly*'.

```
RQLF
qq(claim,
  qq(the,A::{{time,[now]},A},
    qq(few,B::{[den(man)]@@A,B},
      qq(exists,C,{[den(sleep),arg(subject,B)]@@{simple,A},C}))))
QFF
claim(
  the(A::{{time,[now]},A},
    (({[man?0]@@A,B}
      &({([sleep? -1,{subject,B}]@@{simple,A},C)}
        & consistent(({(([sleep ? -1, {subject,D}]@@ {simple,E}),[#, 4, B::[]])}
                        => absurd))))
            => absurd)))
```

Figure 7.27: QFF for '*few men sleep*'.

7.5.6 At most N vs At least N

The way the cardinalities of the two phrases '*at least N*' and '*at most N*' are represented is based on the way cardinality-based calculations are performed within the theorem

prover. For instance, '*at least three men...*' in Figure 7.28, expresses a set of men of cardinality that is equal to 3 or more, thus {card,VAR, >=3}. However, because the size of the set is not actually known, when it comes to answering queries about that set of men, the only answers that can be given are the ones in regard of '*one/two/three men*' or '*at least one/two/three men*'. Contrarily, the set of men in '*at most three men...*' (Figure 7.29), can be of size 3, 2, 1 or even none; 0. So, we can not be certain about any N if it is 3 or less, but what is known for sure is that N **can not be 4 or more**. Therefore, as given in Figure 7.29, the QFF for '*at most three men slept.*' will be a negative rule with cardinality clause {card,D,>=4} which can be read as: *it is absurd to assume that there is a past sleeping event involving 4 or more men*.

```
RQLF
qq(claim,
  qq(exists,(C::{[den(man)]@@A,C}&{card,C,>=3}),
    qq(the,A::{{time,[past]},A},
      qq(exists,B,{([den(sleep),arg(subject,C)]@@{simple,A},B)}))))
QFF
claim(
  (({[man?1]@@A,[#,4]}& {card,[#,4],>=3})
      & (the,A::{{time,[past]},A},
        & {(([sleep?1, {subject,[#,4]}]@@ {simple,A}),[#, 6])})))
```

Figure 7.28: QFF for '*at least three men slept.*'

```
RQLF
 qq(claim,
   qq(predet(most, 3),C::{[den(man)]@@A,C},
     qq(the,A::{{time,[past]},A},
       qq(exists,B,{([den(sleep),arg(subject,C)]@@{simple,A},B)})))
QFF
claim(
  (({[man? -1]@@B,D}& ({card,D,>=4}
    & (the,A::{{time,[past]},A},
      & {([sleep? -1,{subject,D}]@@{simple,B},C)}))=> absurd))
```

Figure 7.29: QFF for '*at most three men slept.*'

7.5.7 Utterance Type Matters

Utterance labels are generally the wrappers of their whole QFFs, as given in (QFF-R1) and shown in all of the above examples.

QFF-R1: QFF of utterance labels
Replace `qq(Utter_label,R)` by `Utter_label(R')` where `R'` is the QFF of `R`.

These labels are still there because they are the indicators of whether a binding variable of a quantificational operator is to be left as it is or to be turned into a term during the removal process. Put differently, in `claim`s, as illustrated above, the variables of positively occurring `exists` and negatively occurring `forall` are generally the ones that get skolemized. The ultimate goal is making inferences, which is trying to find an answer to a `query` from the list of available information (of which `claim`s are a part) to the inference engine. For that to happen, a `query` has to have reverse skolemization decisions. Meaning variables that get skolemized in a `claim` remain variables in the counterpart `query` example and visa versa. We obtain that by reversing the context's polarities, i.e. by setting the initial mark to `-1` during polarity marking of queries (Section 7.5.1). For example, notice the difference in skolemization between '*a man slept .*' and '*a man slept?*' in Figure 7.30. In the `claim`'s QFF, '*man*' and '*sleep*' are positively marked and hence their variables were skolemized, while in the `query` they were marked with `-1` and consequently their variables were not skolemized.

Claim QFF

```
claim(the(A::{{time,[past]},A},
  (({[man?1]@@A,[#,8]}& {card,[#,8],=1})
    & {(([sleep?1, {subject,[#,8]}]@@ {simple,A}),[#, 9])})))
```

Query QFF

```
query(the(A::{{time,[past]},A},
  (({[man? -1]@@A,B}& {card,B,=1})
    & {(([sleep?-1, {subject,B}]@@ {simple,A}),C)})))
```

Figure 7.30: QFF of '*a man slept.*' vs '*a man slept?*'

7.6 Inference Friendly Form (IFF)

Now that we have constructed QFFs, we are a few steps away from obtaining the final form that the inference engine can reason with: IFFs. A QFF that contains REs, not

only DNPs but referential times as well, will first proceed by resolving them to their referents (to be discussed in Section 7.7). Second, the clauses that constitute a QFF are normalized by applying the rules in (IFF-R1). Rule (IFF-R1)(i) ensures that logical symbols linking QFF clauses are distributed in the most convenient way for the inference engine when co-occur. Rule (IFF-R1)(ii) converts rules, as much as possible, into Horn form (i.e.rules with conjoined positive literal antecedents and a single positive literal consequent as in Figure 7.31). That is motivated by the fact that the adopted inference engine (Section 3.3) is implemented in Prolog, and most of its rules of inference runs backward. The third rule, (IFF-R1)(iii), allows modified nominals whose modifiers are intersective (i.e. have `*(+)` as a sign) to have their modifiers detached. For instance, see the clause of '*an Italian woman*' in '*John loves an Italian woman*' (Figure 7.32) before and after normalizing. Finally, rule (IFF-R1)(iii) is the double negation elimination rule.

IFF-R1: QFF clauses normalization

For clauses `R, P` and `Q` in their QFF:

(i) **Implication distribution over conjunction:** Turn `R => (P & Q)` into `(R => P) & (R => Q)` and then reapply the rules.

(ii) **The conjunction of the antecedents:** Turn the nested implications `(R => (P => Q))` into `((R & P)=> Q)` and then reapply the rules.

(iii) **Detach intersective modifiers:** Turn {`R, X`} (where `R` is a modified nominal) into {`P,X`} `&` {`Q, X`} (where `P` is the modified nominal and `Q` is the modifier).

(iv) **Double negation elimination:** Turn `((R => absurd) => absurd)` into `R`.

```
QFF
claim(the(A::{{time,[now]},A},
  ({[man? -1]@@A,E}
    => ({[woman? -1]@@A,D}
      => {(([love?1, {dobj,D}, {subject,E}]@@ {simple,A}),[#, 8, D, E])}))))

IFF
claim({{time,[now]},[#,1]},
  (({[man? -1]@@[#,1],A}& {[woman? -1]@@[#,1],B})
    => {(([love?1, {dobj,B}, {subject,A}]@@ {simple,[#,1]}),[#, 8, B, A])})))
```

Figure 7.31: Example of the application of rule (IFF-R1)(ii).

```
QFF
claim(the(A::{{time,[now]},A},
  (({(([woman?1,modifier(amod(simple),*(+),[Italian?1])]@@A),[#,8])}
    & {card,[#,8],=1})
      & the((B :: ({{name,[den(John)]}@@A,B}& {card,B,=1})),
        {(([love?1,{dobj,[#,8]},{subject,B}]@@ {simple,A}),[#, 9])}))))
IFF
claim({{time,[now]},[#,1]},
  (({[woman?1]@@[#,1],[#,8]}
    & {[Italian?1]@@[#,1],[#,8]})
      & {card,[#,8],=1})
        &({{name,[John?1]}@@A,[#, 10])}& {card,[#,10],=1})),
          {(([love?1,{dobj,[#,8]},{subject,[#,10]}]@@{simple,[#,1]}),[#,9])})))))
```

Figure 7.32: Example of the application of rule (IFF-R1)(iiii).

Doing the above turns a QFF into its IFF. For a `query`, as illustrated in Figure 7.1, that IFF is sent to the inference engine to find its answer. In the case of a claim, the IFF is broken down into facts and/or rules, then asserted as part of the engine's list of avilable information. The process of obtaining these clauses is as indicated in (IFF-R2).

IFF-R2: Obtaining facts/rules form claim's IFF

After discarding the `claim()` wrapper as at it is no longer significant, the conjuncted clauses will be recursively broken down into a list of conjuncts: `P` and/or `(A=>C)`, such that:

(i) The clause `P` is a fact, iff it is an atomic clause (i.e. does not contain the `&` nor the `=>` symbols).

(ii) The clause `(A=>C)` is a rule, iff `C` is atomic clause (as (IFF-R1)(i) should have been already applied).

Each of these final clauses will be asserted in a certain format, see Section 7.6.2 and 7.6.1, with a feature called `STATUS`. Figure 8.6 shows the `STATUS` possible values which are: `real`, `assimilated`, `claimed` or `temp`. This classification is used by the inference engine to determine which of the stored information and when can be used during a proof process. Clauses derived from a `claim`'s IFF which are not obtained by assimilation during dereferencing referential terms always have `claimed` as their status, while clauses obtained during dereferencing DNPs can be either `claimed`, once they are resolved, or `assimilated`; if assumed (see Section 7.7). The discussion of the remaining types and the differences among all of the four in detail will be better left

until Chapter 8.

7.6.1 Facts

As indicated above, facts are atomic clauses that when asserted have the following format: `fact(Clause,STATUS)`. Among the atomic clauses, equalities which are of the form ({`SF0=SF1`}@@{`aspect, SF2`}) have a different assertable structure which is a tuple: `eclass(A,B,C,STATUS)` and need to follow a certain procedure before being asserted. The procedural steps aim at avoiding repetition and running into vicious circles. The field `C` is always the same as the context clause (`CTXT` henceforth) {`aspect, SF2`}. And it is important for it to be part of the equality tuple because equalities are time-stamped in the same way that ordinary events are (e.g. '*John **is** a fool*' is different from '*John **was** a fool*' in a scene that we cannot infer one from the other) However, `A` and `B` in the `eclass` are not necessarily the same as `SF0` and `SF1` respectively, as each should be:

- ***De-referenced*** to the last value of its chain of equals in the current context `C`. For example, if `(SF0=X, X=Y, Y=A` $\rightarrow$ `SF0=A)` and we say that `SF0` was de-referenced to `A`. Then,
- ***Sorted*** so that `A` represent the lexically greatest of the two and `B` is the other one.

In Figure 7.34, the terms in the `eclass` tuple have the same `SF`s as the ones in the equality clause: ({`[#,4]=[#,6]`}@@{`simple,[#,3]`}) but the reverse order, and there are no other entries in a list of equalities so no need for de-referencing. However, consider having the list of `eclass`es shown in Figure 7.33. Now if an `eclass` for the clause

```
** EQUALITIES **
[eclass([#, 5], [#, 2], {simple,[#, 1]}, claimed),
eclass([#, 7], [#, 4], {simple,[#, 1]}, claimed),
eclass([#, 4], [#, 3], {simple,[#, 1]}, claimed)]
```

Figure 7.33: Example of de-referencing an equality argument.

({`[#,5]=[#,7]`} @@{`simple,[#, 1]`}) is to be added, instead of asserting it immediately as `eclass([#, 7], [#, 5], {simple,[#, 1]}, claimed)`, both `[#, 5]` and `[#, 7]` have entries in the `EQUALITIES` list, and hence must be de-referenced. The argument `[#, 5]` is only equal to `[#, 2]` and `[#, 2]` is not equal to anything else, thus, `[#, 5]` is de-referenced to `[#, 2]`. On the other hand, the argument `[#,7]` is de-referenced

to [#, 3], because the latter is equal to [#, 4] and [#, 4] itself is equal to [#, 3]. Consequently, the eclass to be added is eclass([#, 3], [#, 2], {simple,[#, 1]}, claimed).

```
IFF
claim(({{time,[past]},[#,3]}
  & (({[fool?1]@@[#,3],[#,4]}& {card,[#,4],=1})
    & ({({name,[John?1]}@@A,[#, 6])}& {card,[#,6],=1}),
      {[#,4]=[#,6]}@@{simple,[#,3]})))
** FACTS **
[fact({card,[#,4],=1}, claimed),
fact({[fool?1]@@[#,3],[#,4]}, claimed),
fact({{time,[past]},[#,3]}, claimed),
fact({card,[#,6],=1}, assimilated),
fact({{name,[John?1]}@@A,[#,6]}, assimilated)]
** EQUALITIES **
[eclass([#, 6], [#, 4], {simple,[#,3]}, claimed)]
```

Figure 7.34: An extracted facts and equalities from '*John was a fool.*'

7.6.2 Rules

In a rule (A ⇒ C), A is the antecedent and C is the consequent. Before asserting a rule, its antecedent and consequent are checked to determine whether a rule is to be applied Forward or backwards. A rule is forward and is added as (A ⇒ C: STATUS) if its antecedent is an atomic clause and is not a consistent condition and its consequent is not absurd. A rule is backward, otherwise, and it is asserted as (C ⇐ A: STATUS). See Figure 7.35 and Figure 7.36 for examples of forward and backward rules, respectively.

```
IFF
claim(the([#,2]::{{time,[now]},[#,2]},
  ({[man? -1]@@[#,2],A}
    => {(([sleep?1, {subject,A}]@@ {simple,[#,2]}),[#, 1, A])})))

** FACTS **
[fact(time,[now],[#,2], assimilated)]
** FORWARDS RULES **
[({[man? -1]@@[#,2],A}
  => {[sleep? 1,{subject,A}]@@{simple,[#,2]},[#,1,A]}:claimed)]
```

Figure 7.35: Extracted facts and rules form '*every man sleeps.*'

```
IFF
claim(the([#,4]::{{time,[now]},[#,4]},
  (({[man]@@[#,4],A}& {[car]@@[#,4],B})
    => {(([love, {dobj,B}, {subject,A}]@@ {simple,[#,4]}),[#, 3, B, A])})))

** FACTS **
[fact({{time,[now]},[#,4]}, assimilated)]
** BACKWARDS RULES **
[({[love,{dobj,A},{subject,B}]@@{simple,[#,4]},[#,3,A,B]}
  <= {[man]@@[#,4],B}&{[car]@@[#,4],A}:claimed)]
```

Figure 7.36: Extracted facts and rules form '*every man loves every car.*'

7.7 Resolving Referring Expressions

The previous section has shown that the process of turning QFFs into IFFs involves resolving REs. However, we opted to postpone the discussion of how REs are actually resolved until now, as the previous section explains some of the background machinery that is used here.

Up until this point, REs have been turned into terms of the following pattern: `the (VAR:: bound restrictor)`, which is their RQLF. They were kept in their RQLF and were not fully processed in QFF, because in contrast to other quantificational operators, their binding variables can not just be either skolemized or left as they are, instead, they need to be resolved. Resolving a RE means finding its referent. Hence, as explained in Section 5.3.1 and illustrated in Figure 7.37, this work's way of finding a RE's referent starts with attempting to prove that there is only one thing in the list of available information that satisfies its descriptor (its bound restrictor). This process is called *anchor*ing. If **the** referent was found, an instance of its SF is going to be assigned to the RE's binding variable, `VAR`, and all of its occurrences in the QFF. Nonetheless, if it was not found, which is the case when a RE is first mentioned in a conversation, the RE is to be *assimilate*ed. When a RE is assimilated, it means that the existence of its referent has been assumed and, thus, its binding variable to be assigned a new SF. This variable assignment will make all of its occurrences in the QFF be instantiated to that new SF as well. After that, the resolved RE by assimilation is asserted as a fact, with `assimilated` as its `STATUS`.

A running example for resolving REs is given in Figure 7.38 for the discourse in (7.16), in which '*John*' in p_2 is assimilated to the new constant `[#,50]`, while '*the*

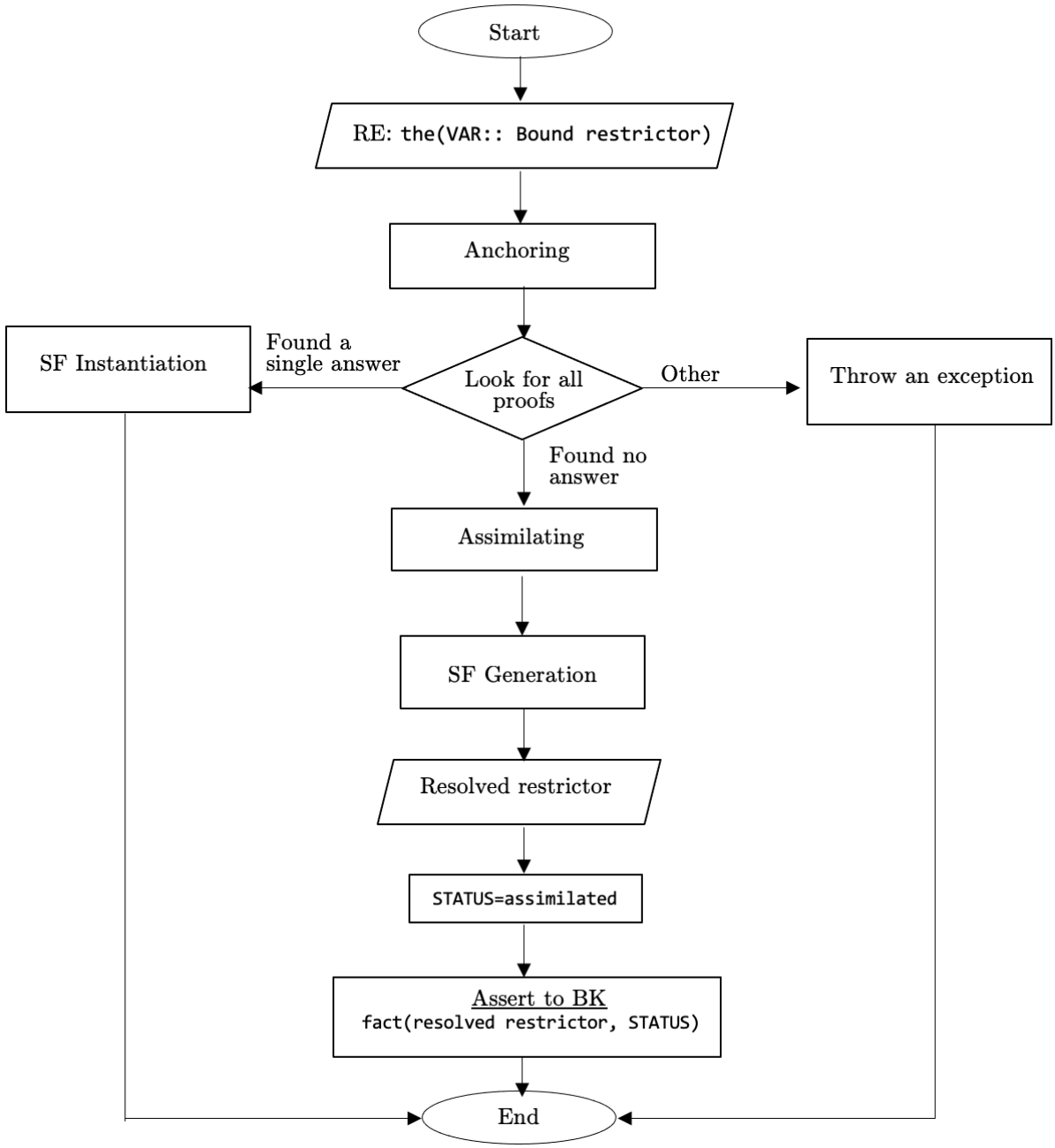


Figure 7.37: Resolving REs general flow chart.

woman' is dereferenced to the existentially quantified '*woman*' in p_1, and thus instantiated to its SF, `[#,7]`. This makes the '*love*' event involving these two REs, '*John*' and '*the woman*', as follows: `{[love?1,{dobj,[#,50]},{subject,[#,7]}]...}`. The pronoun '*He*', on the other hand, was resolved to '*John*' as the only single male mentioned in the conversation, and thus its SF will be like John's which is `[#,50]` and so

the subject of the '*died*' event; {[die?1,{subject,[#,50]}]...}

(7.16) p_1. A woman slept.
p_2. The woman loves John.
p_3. He died.
...

```
** FACTS **
[fact({[die?1,{subject,[#,50]}]@@{simple,[#,9]},[#,111]}, claimed),
fact({[male?1]@@A,[#,50]}, claimed),
fact({([love?1,{dobj,[#,50]},{subject,[#,7]}]@@{simple,[#,49]},
[#, 48])},
claimed),
fact({card,[#,50],=1}, assimilated),
fact({{name,[John?1]}@@B,[#,50]}, assimilated),
fact({{time,[now]},[#,49]}, assimilated),
fact({[sleep?1,{subject,[#,7]}]@@{simple,[#,9]},[#,8]}, claimed),
fact({card,[#,7],=1}, claimed),
fact({[woman?1]@@[#,9],[#,7]}, claimed),
fact({{time,[past]},[#,9]}, assimilated)]
```

Figure 7.38: List of extracted facts from (7.16).

Till now, the discussion of resolving REs has been rather general and theoretical. Because resolving REs by anchoring is actually about attempting to prove that there is exactly one thing which can be shown, using the information available to the participants, to satisfy the restrictor, many aspects of their resolution are to be re-visited in the next chapter when the inference engine is explained. One aspect is regarding pronouns. In the previous examples the pronoun '*he*' has been resolved to the name '*John*' assuming that it satisfies its description which is the singular salient male. However, nothing in John's representation mentions the fact of him being a male, but the inference engine managed to make him the referent of '*he*' by making use of some *hand-coded* general information, in the list of avilable information, that interprets John as a singular male. The other aspect is about natural logic *subsumption* relations; as it allows a RE of the kind '*The NP* $_1$' to be resolved to NP_2 if NP_1 subsumes NP_2. For instance in (7.17), '*the man*' in p_2 is dereferenced to '*an old fat man in the park*' in p_1 because '*old fat man in the park*' $\leq$'*man*'.

(7.17) p_1. There is an old fat man in the park.
p_2. The man died.
...

Lastly, in Section 7.3.5, we have mentioned that RCs pronouns and implicative complements' zero subjects are to be resolved at this point. In contrast to other REs, the referents for these two kinds of expression are easily identified. For the relative pronoun, it is the noun that the whole RC modifies, while for implicative clause's zero subject it is the same subject as the containing main clause. However, because their referents might need resolving themselves, we had to leave their resolution till this point, such that, in '*the man who loves Mary died*' for instance, once the definite NP that a RC modifies has been resolved, the relative pronoun is resolved by instantiating its SF.

7.8 Summary

The inference engine we are using in this work consists of a handful of inference rules. To ensure the applicability of these rules, we turn decorated DTs into a suitable form. Therefore, in this chapter we have discussed the transformational steps that we have designed (CO2.2) to turn DTs into IFFs. These include:

- Turning specifiers into bound operators named `qq` forms, that in turn convert DTs into QLFs.
- Ensuring a preferred reading for the main utterance by resolving the QLF.
- Turnimg RQLFs into QFFs by removing quantifiers and turning their variables into terms where applicable, and marking the form with polarities.
- Resolve any REs in a QFF and normalize them into assertable IFFs: lists of facts and/or rules.

Chapter 8

Constructive SATCHMO+

As Figure 1.2 (repeated here as 8.1) shows, inputs from a discourse were parsed in Chapter 6 and turned into IFFs in Chapter 7. This chapter is dedicated to the third and final part of our inferential system, the inference engine itself, named CSATCHMO+ (marked in black in Figure 8.1). In Chapter 4, we learned the key features of incorporating semantic containment ($\leq$) and polarity marking to design (CO3) a matching algorithm on trees. This chapter explains how the natural logic matching algorithm works and its contribution (CO5.1) to the inference process (Section 8.4). We have also learned several important aspects regarding defaults and attitude verbs in Chapter 5 to design (CO2.3) appropriate rules to handle them and extend (CO5.2) the inference engine to include them. Here (Section 8.5), we show through examples how reasoning about these constructions is carried out using those designed rules. Other than that, the first part of this chapter will explain the engine's proof algorithm (Section 8.2) and the different kinds of information that are available for it to use (Section 8.3), and the last part of it discusses the engine's performance (Section 8.6).

8.1 The Engine's Data-Flow

The inference engine explained here is named **C**SATCHMO**+**. It is an adaptation of Ramsay's basic **constructive** SATCHMO program, Section 3.3, but for normalized DTs and with two different **extensions**. The first extension handles inferences about certain constructions (defaults and attitudes) as a means of inference rules. The second replaces Prolog's straight unification[1] with a matcher that is implemented on the basis of natural logic's containment relation (explained in Chapter 4). These extensions will be discussed later on in this chapter. For the time being, a view of the engine's

[1] In Prolog, two terms unify if both or one of them is a variable, or are identical.

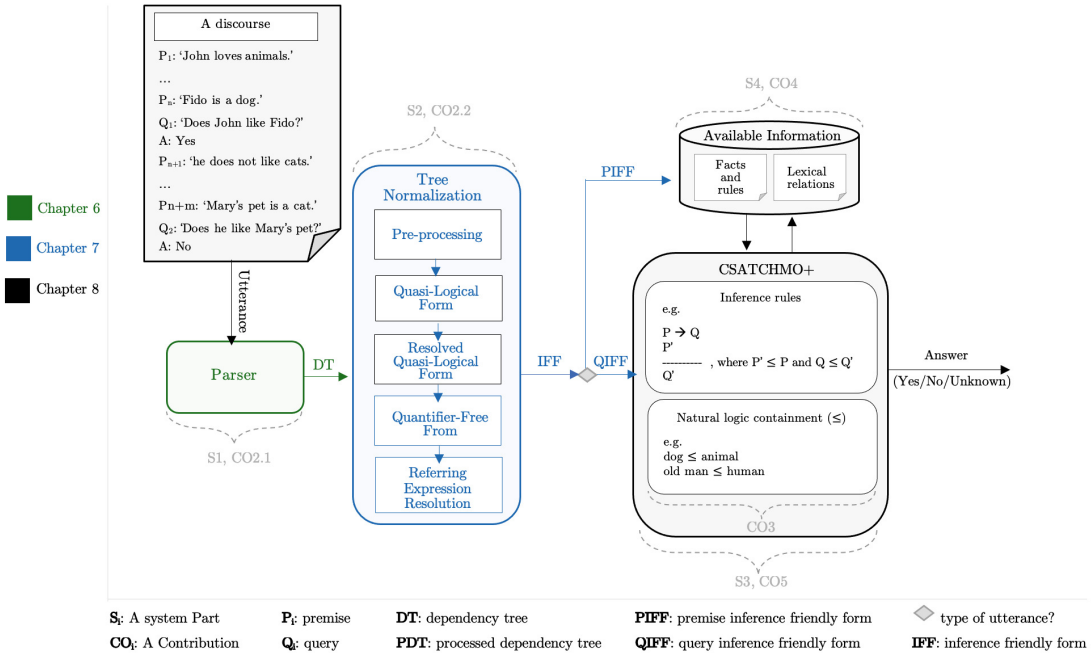


Figure 8.1: Data-flow through the inferential system.

general data-flow will be given and the way that basic proofs are conducted will be illustrated. As Figure 8.2 shows, the engine receives a normalized query, `NQ`[2], as an

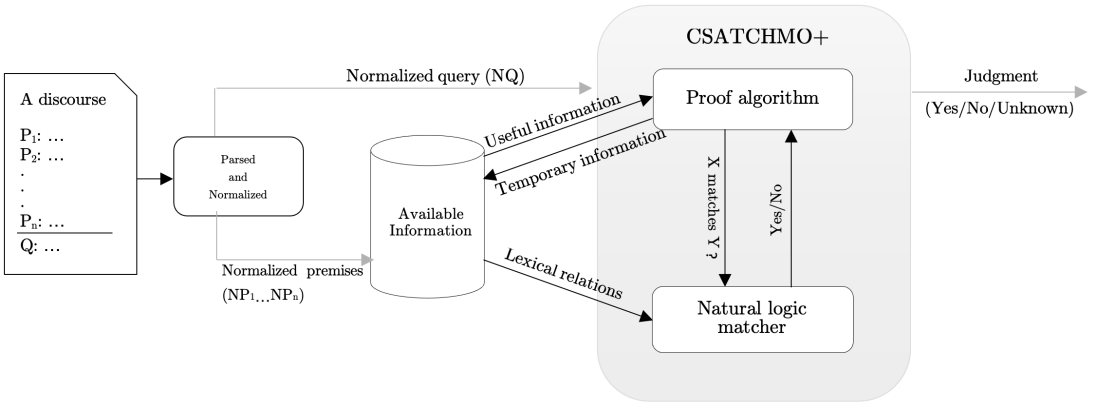


Figure 8.2: Data-flow from and to CSATCHMO+.

input and produces as an output the answer to that query. CSATCHMO+ is a three-way classification engine. Therefore, the answers it gives are either `Yes`, `No` or `Unknown` for entail, contradict, and neutral respectively. As indicated by the (JUDGEMENT) rules, the answer is `Yes` when the engine is able to prove `NQ`; `No`, if the engine cannot not

[2]In this chapter, `NQ` will be used as an abbreviation for 'normalized query', i.e. a query `Q` that has been parsed then transformed into an IFF: `NQ = IFF(DT(Q))`

find a proof of NQ, but can for its negation, NQ ⇒ absurd; and unknown if the engine cannot not prove NQ or its negation. In some special cases, which we will discuss later, an answer may have a follow-up comment declaring or stating the condition(s) under which it is to be accepted.

The engine's process of finding a query's answer is mainly the job of the proof algorithm, Section 8.2. During a proof process, the proof algorithm relies on its rules of inference and any useful information that is present in the background at the time, Section 8.3. Moreover, the proof algorithm frequently requires the aid of the natural logic matcher to decide whether two non-identical items are to be considered a match or not. In addition, during the proof construction process, the engine may add any temporarily generated information to the background knowledge in order use it as well for that particular proof.

(JUDGEMENT): CSATCHMO+ possible answers
To find a query's answer, CSATCHMO+ will attempt to prove its NQ such that:
(J-1) If a proof of NQ is found, then the query's answer is 'Yes'.
(J-2) If a proof of NQ's negation i.e. NQ ⇒ absurd is found, then the query's answer is 'No',
(J-3) If no proof is found for NQ or its negation, then the answer is 'Unknown'

8.2 Proof Algorithm

The algorithm, outlined in (PROOF ALGORITHM), is adapted from Ramsay's basic constructive SATCHMO algorithm, with two significant extensions:

- We use *match*ing (Section 8.4) instead of unification for matching goals to facts and rules.
- We split rules into three groups:

  Backwards rules: these are standard Horn clauses with compound antecedents, i.e. rules of the form A1 & ...& An => C

  Split rules: these are, as with the original version of Satchmo, rules with disjunctive consequents, i.e. rules of the form A1 & ...& An => C1 or C2

  Forwards rules: these include Horn clauses with atomic antecedents and rules

expressing higher-order relations. Running Horn clauses with atomic antecedents forwards is simply a matter of convenience. Rules for dealing with higher-order relations are discussed in Section 8.5

(PROOF ALGORITHM)

A proof of a goal `NQ` from a set of facts `FACTS`, forwards rules `F-RULES`, backward rules `B-RULES` and split rules `S-RULES` is carried out in three stages:

(FI) Forwards inference: start by adding the recursive closure of the forward rules to the facts: if `A => C` is in `F-RULES` and `A` can be proved backwards from `FACTS` and `B-RULES`, add `C` to `FACTS`

(BI) Backwards inference: `NQ` can be proved backwards if it:

- (BI-1) matches a fact `NQ'`,
- (BI-2) matches the consequent of a backward rule `(NQ' ⇐ A)`, and the antecedent `A` can be proved.
- (BI-3) is a conjunction `(NQ1 & NQ2)` and both `NQ1` and `NQ2` can be proved,
- (BI-4) is a disjunction `(NQ1 or NQ2)` and either `NQ1` or `NQ2` can be proved,
- (BI-5) is of the form `(A⇒C)`, in which case it can be proved from `<FACTS, F-RULES, B-RULES, S-RULES>` if `C` can be proved from `<FACTS ∪ {A}, F-RULES, B-RULES, S-RULES>` (conditional proof). This rule is particularly important for proving negations, since we treat `not(P)` as a shorthand for `P => absurd`.

(SR) Split rules: `NQ` can be proved if there a split rule `A => (C1 or C2)` where `A` can be proved backwards and `<FACTS ∪{C1}, B-RULES, F-RULES, S-RULES>` and `<FACTS ∪{C2}, B-RULES, F-RULES, S-RULES>` both support proofs of `NQ`.

To demonstrate how basic proofs are constructed using the rules given above, consider the argument in (8.1). The IFFs for the premise (p_1) and the query (q_1) are all given in Figure 8.3.

(8.1)

p_1.	A man slept in the park.
q_1.	Did a man sleep? [Yes]
q_2.	Did no man sleep? [No]

The engine's answer for the query is (`Yes`). That answer is based on the proof given in Figure 8.4. Each proof step is marked with two coloured labels: `LABEL1` which marks the clause to be proven and that helps with navigating through sub-proofs, and

LABEL2 which indicates which of the PROOF ALGORITHM's rules is used to prove that clause. Since the tense clause is already resolved, the first proof step in Figure 8.4 is an attempt to prove the conjunctive clause c consisting of two conjuncts. Thus, the applied rule is BI-3 where each of c 's conjuncts is proved separately. The first conjunct of the two is c1 and is a conjunctive clause itself, hence the engine applies rule BI-3 to prove each of its constituents c1.1 and c1.2. The first matches the claimed fact {[man? 1]@@[#,9],[#,7]}, i.e. it is proved using rule BI-1. The same goes for the second conjunct, c1.2, as it matches {card,[#,7],=1}. The final atomic clause, c2, also matches the claimed fact {[sleep? 1,...} and hence is said to proved using rule BI-1 as well. After the end of every successful sub-proof, there is a line, including the label of the proved clause, that states so. For instance, the line (**Proof of fact c1 succeeded**) indicates that the CSATCHMO+ managed to prove c1. A successful proof of all parts of c means a successful proof of c, and hence the answer is Yes.

A proof of c would be unsuccessful if at least one of the sub-proofs did not go through. In such cases, before giving an answer, CSATCHMO+ would further investigate by attempting to prove the negation of c. For example, consider answering (q_2) from the same p_1 in (8.1). As illustrated in Figure 8.5, CSATCHMO+ has tried every possible way for getting a proof of the query c by using all the applicable rules from the (PROOF ALGORITHM). After failing to prove c, the engine attempts to prove its negation c′ (note that negating c will invite the application of the double negation elimination rule given in (IFF-R1)(iii), and hence c′ is a simple conjunctive clause). As the proof in Figure 8.5 shows, the sub-proofs of all of c′ constituents were all successful, and so the proof of c′ is complete. Finding a proof of c′ means that the answer for the original query is No.

```
** FACTS **
[fact({(modifier(ppmod(+), *(+), [in?1,{ppcomp,[#,10]}]@@{simple,[#,9]}),
[#, 8])},claimed),
fact({[sleep?1,{subject,[#,7]}]@@{simple,[#,9]},[#,8]}, claimed),
fact({card,[#,7],=1}, claimed),
fact({[man?1]@@[#,9],[#,7]}, claimed),
fact({card,[#,10],=1}, assimilated),
fact({[park?1]@@A,[#,10]}, assimilated),
fact({{time,[past]},[#,9]}, assimilated)]
```

```
**QUERY1**
query(the([#,9]::{{time,[past]},[#,9]},
  (({[man? -1]@@[#,9],A}& {card,A,=1})
    & {(([sleep?-1, {subject,A}]@@ {simple,[#,9]}),B)})))
**QUERY2**
query((((the([#,9]::{{time,[past]},[#,9]},
  {[man?1]@@[#,9],[#,11]}& {card,[#,11],A})
     & {(([sleep?1, {subject,[#,11]}]@@ {simple,[#,9]}),[#, 12])})
       => absurd))
```

Figure 8.3: The IFFs of the premise and query of the argument in (8.1)

```
** PROOF **
Trying Horn proof of conjunction c BI-3:
{[man? -1]@@[#,9],A}& {card,A,=1})
 & {(([sleep?-1, {subject,A}]@@ {simple,[#,9]}
  Trying Horn proof of conjunction c1 BI-3:
  ({[man? -1]@@[#,9],A}&{card,A,=1}
    Looking for a fact that matches c1.1 BI-1:
    {[man? -1]@@[#,9],A} matches a claimed fact {[man? 1]@@[#,9],[#,7]}
    Proof of c1.1 succeeded
    Looking for a fact that matches c1.2 BI-1:
    {card,[#,7],=1} matches a claimed fact {card,[#,7],=1}
    Proof of c1.2 succeeded
  Horn proof of conjunction c1 succeeded
  Looking for a fact that matches c2 BI-1:
  {[sleep? -1,{subject,[#,7]}]@@{simple,[#,9]},A} matches a claimed fact
  {[sleep? 1,{subject,[#,7]}]@@{simple,[#,9]},[#,8]}
  Proof of c2 succeeded
Horn proof of conjunction c succeeded:
Answer: Yes
```

Figure 8.4: CSATCHMO+ proof for the argument (8.1).

```
** PROOF **
Trying Horn proof of c:
(({[man?1]@@[#,9],[#,11]}&{card,[#,11],A})
  &{[sleep?1,{subject,[#,11]}]@@{simple,[#,9]},[#,12]})=>absurd

Looking for a fact that matches c BI-1:
Unsuccessful
Looking for a rule whose LHS matches c BI-2:
Unsuccessful
About to try conditional proof of c BI-5:
Trying to prove c by asserting c1:
({[man?1]@@[#,9],[#,11]}&{card,[#,11],A})
  &{[sleep?1,{subject,[#,11]}]@@{simple,[#,9]},[#,12]}
And trying to prove c2: absurd
   Looking for a fact that matches c2 BI-1:
   Unsuccessful
   Looking for a rule whose LHS matches c2 BI-2:
   Unsuccessful
  Tried every way of getting a Horn proof of c2
Tried every way of getting a Horn proof of c

Trying to answer the contrary c/:
Trying Horn proof of conjunction c/ BI-3:
{[man? -1]@@[#,9],A}&{card,A,B}
  & {([sleep? -1,{subject,A}]@@{simple,[#,9]},C)}
  Trying Horn proof of conjunction c1/ BI-3:
  {[man? -1]@@[#,9],A}&{card,A,B}
    Looking for a fact that matches c1.1/ BI-1:
    {[man? -1]@@[#,9],A} matches a claimed fact {[man? 1]@@[#,9],[#,7]}
    Proof of a fact c1.1/ succeeded
    Looking for a fact that matches c1.2/ BI-1:
     {card,[#,7],A} matches a claimed fact {card,[#,7],=1}
    Proof of a fact c1.2/ succeeded
  Horn proof of conjunction c1/ succeeded
  Trying Horn proof of conjunction c2/ BI-3:
  {[sleep? -1,{subject,[#,7]}]@@{simple,[#,9]},D})
  Looking for a fact that matches c2/ BI-1:
  {[sleep? -1,{subject,[#,7]}]@@{simple,[#,9]},D} matches a claimed fact
  {[sleep?1,{subject,[#,7]}]@@{simple,[#,9]},[#,8]}
  Proof of a fact c2/ succeeded
Horn proof of conjunction c/ succeeded
Answer: No
```

Figure 8.5: CSATCHMO+ proof of q_2 from (8.1).

8.3 Available Information

In addition to the supplied premises theorem provers quite often require the aid of some pre-existing knowledge to support their proofs [Bos and Markert, 2005]. Take, for example, the query (q) in (8.2): no answer can be derived from (p_1) alone, unless there

is the prior knowledge that '*every dog is an animal*'. In everyday human conversation, the existence of such common information is not necessarily explicit; it is in the minds of the conversation participants. Therefore, in addition to the information conveyed by an inference problem's premises, inference engines generally require any other requisite knowledge, supplied either manually or automatically, from external sources as their *background knowledge* (BK).

(8.2)
p_1. Fido is a dog.
---
q. Is Fido an animal?

The information available to CSATCHMO+, as Figure 8.6 shows, can be split into three parts. The first part, and the only one discussed so far, comes from the IFFs that have been constructed and asserted (or assimilated) from an inference problem's premises (i.e. `claims`): these are labelled as the *minutes*. The second group includes general information, such as properties associated with proper names (e.g. '*John*' is typically held by a singular male and '*Mary*' by a singular female), lexical relations (e.g. dog $\leq$animal, man $\leq$human, love $\leq$like) that are available in dictionaries such as WordNet, and a range of hand-coded rules. This group is labelled *Real* and is discussed further in Section 8.3.1. Lastly, there is *temporary* information (Section 8.3.2), which is anything asserted during a course of a conditional proof.

Not all of the information available to CSATCHMO+, then, is permanent. Real information, i.e. the system's knowledge of word meanings, is permanent. Temporary information is introduced during conditional proofs, and is retracted as soon as the proof either succeeds or fails. The minutes are a record of what has been accepted during the course of a single continuous discourse (i.e. series of claims and questions). In contrast to temporary information, we do not discard the minutes' contents after the end of one proof, as we allow a single discourse to contain a series of claims and queries. Therefore, the minutes' content is discarded only when a new conversation has started.

All of the above kinds of information are asserted in the format of IFFs. They differ only with respect to their `status` value which is either `real`, `claimed`, `assimilated` or `temp(SF)`. That value is important to indicate their source and hence their validity for for use by the engine. Since constructing the minutes has been explained at length in Chapter 7, the following two sections will focus more on the other two types of information: real and temporary.

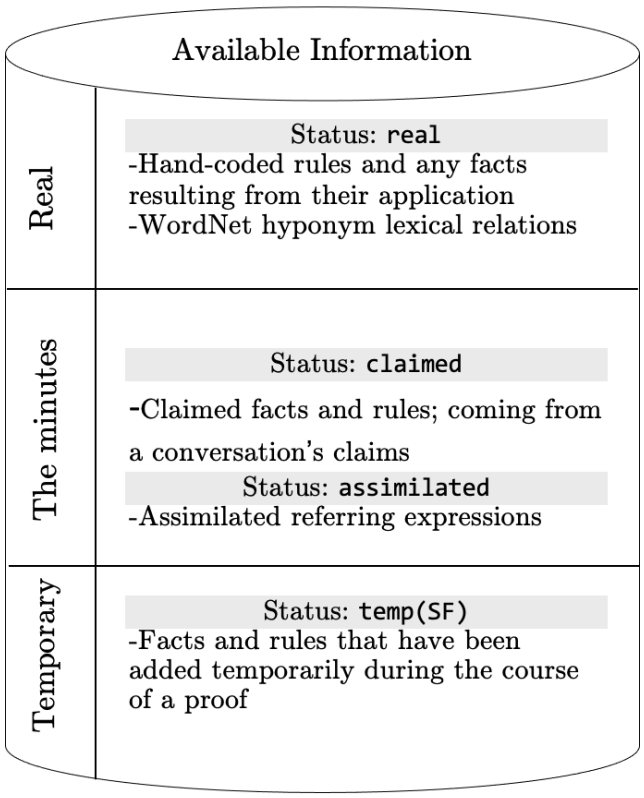


Figure 8.6: The classification of the background knowledge entries and their STATUS.

8.3.1 Real Information

WordNet Lexical Relations

Lexical relations are established links between words with respect to their meaning. The most common types of lexical relations are: *synonyms* such as (fast/quick), *antonyms* such as (small/big), and *hyponyms* such as (dog/animal) and (walk/move).

A well-known source of such relations is the manually constructed and electronically published lexical database of English; WordNet[3]. Among the different lexical connections that WordNet offers,in this work we are particularly interested in hyponym relations. To make WordNet hyponym relations easily accessible, we have extracted

[3]WordNet "is a large electronic lexical database of English" [Fellbaum, 2010, p. 231] that was manually constructed in 1986 at Princeton University and continually maintained and updated. It is designed as a semantic network of words and phrases interlinked by means of relations. The building block for WordNet is sets, called *synsets*. A synset is an unordered set of words or phrases that share the same concept, i.e. synonyms. Aside from synonyms, other relations that show the interconnections between synsets include antonyms, hyponyms and troponyms. Synsets are spread over the syntactic parts of WordNet (*verbs*, *nouns*, *adverbs* and *adjectives*) and most of them are labelled with domain names, such as {medical, sport, etc.}

these relations and built up a table of all specific word-general word relations under the name *hyponyms* table. The table consists of factual tuples of the form `hyp(Specific, General, POS)`, see Figure 8.7 for examples. This table is used by the matcher rather than directly accessed by the theorem prover itself. Put differently, in course of a proof, the engine might ask the matcher if two terms are considered a match, and the matcher will make use of the hyponyms table when answering.

```
hyp('man','human',n),
hyp('dog','animal',n),
hyp('love','like',v),
hyp('walk','move',v)
...
```

Figure 8.7: Sample from the hyponyms table.

Hand-Coded Rules

Hand-coded rules are commonly about general knowledge that one is expected to have, but is not present in WordNet. What we hand-coded into the engine's BK includes useful properties about the holders of proper names and entailment patterns of attitude verbs. Examples of the first is shown in Figure 8.8. These rules are particularly important to derive facts that are (as seen in Section 7.7) useful for resolving REs. For instance, asserting the IFF of '*John loves Mary*' will invoke the application of the forward rules in Figure 8.8, and hence add the resulting new facts '*male*' and '*female*' to the list of facts in Figure 8.9. Doing so will make resolving, for example, the pronoun '*he*' in '*he died*' (whose descriptor is: `the(X::{[den(male)]@@T,X} &{recent,X} &{card,X,=1})`) attainable.

```
 {{name,['John'?_]}@@T,X} => {[male?1]@@T, X},
 {{name,['Mary'?_]}@@T,X} => {[female?1]@@T, X},
...
```

Figure 8.8: Examples from the simple hand-coded rules.

The second group of hand-coded rules is about the entailment properties of attitude verbs. Given the general structure of these verbs and their complements `{xcomp...}` in Figure 8.10, the entailment pattern of `Att_verb` is `[PP/EP,PN/EN]` and is already marked in the dictionary of the grammar (Section 6.2.3). The polarity of its context is indicated by `(Pol)` and is marked in the QFF (Section 7.5). Therefore, the hand-coded

```
** FACTS **
[fact({[female?1]@@A,[#,6]}, claimed),
fact({[male?1]@@B,[#,5]}, claimed),
fact({([love?1,{dobj,[#,6]},{subject,[#,5]}]@@{simple,[#,4]},[#, 3])},
claimed),
fact({card,[#,6],=1}, assimilated),
fact({{name,[Mary?1]}@@C,[#,6]}, assimilated),
fact({card,[#,5],=1}, assimilated),
fact({{name,[John?1]}@@D,[#,5]}, assimilated),
fact({{time,[now]},[#,4]}, assimilated)]
```

Figure 8.9: The directly and indirectly obtained facts from '*John loves Mary*'.

rules ensure that the entailment decision about the proposition `Prop` in the attitude complement obeys this marked information. For example, the rule in Figure 8.11 states that when an attitude verb is positively asserted and its pattern unifies with `[_/ 1, _/_]`, the embedded proposition `Prop` can be entailed. A further illustration of attitude rules and their application in proofs is given in Section 8.5.2.

```
fact({(([Att_verb? Pol,
          {xcomp([PP/EP,PN/EN]), Prop}, ...]@@ {Aspect,SK0}),SK1)},claimed)
                 entailment pattern
```

Figure 8.10: The general structure of attitude verbs.

```
{[Att_verb? 1, {xcomp([_/ 1, _/_]), Prop},...]@@CTXT, X} => Prop
```

Figure 8.11: Example of an attitude rule.

There is, of course, a vast amount of other general knowledge that participants in a conversation are assumed to have – that after someone buys something they own it, that if two people get divorced they are not married, that if you are allergic to something you should avoid eating it, etc. We concentrate here on lexical relations and propositional attitudes because they involve making changes to the way the inference engine works, whereas rules such as Figure 8.12 can be handled by the standard version. Such rules are crucial to understanding general conversation, but since they do not require any extensions to the inference engine we will not say more about them here.

```
{[buy, {dobj, D}, {subject, S}]@@{A1, T0}, E} & {T1 > T0}
        => exists(X, {[own, {dobj, D}, {subject, S}]@@{A2 ,T1}, X})
```

Figure 8.12: If S buys D at T0 then at some later time T1 S owns D.

8.3.2 Temporary Information

The final kind of information the engine can use during proofs is the temporary information. As mentioned above, it gets added during a conditional proof course and immediately discarded when that proof ends. To keep track of the temporary information that belongs to a particular proof and to ensure the removal only of intended assertions, they are assigned a unique temporary status `temp(SF)`. Put differently, it is possible that during a conditional proof another sub-conditional one starts, so their temporarily asserted information will have different values for `SF` and hence different status. Lastly, like any other added fact, adding temporary information will invite any relevant forward rule application. Any resulting information will have the same temporary status as the assertion(s) that caused their introduction, such that they get removed at the end of the conditional proof as well. To give an example, consider the proof of '*did no man sleep?*' depicted in Figure 8.5. One of the proof attempts was a conditional proof where the head of the negation rule was added temporarily to the list of available information and the consequent `absurd` was proved. This temporarily added information can be seen in the mid-proof version of the list of facts and rules, given in Figure 8.13.

```
** FACTS **
[fact({[sleep?1,{subject,[#,11]}]@@{simple,[#,9]},[#,12]},temp(temp225347)),
fact({card,[#,11],A},temp(temp225347)),
fact({[man?1]@@[#,9],[#,11]},temp(temp225347)),
fact({(modifier(ppmod(+), *(+), [in?1,{ppcomp,[#,10]}]@@{simple,[#,9]}),
[#, 8])},claimed),
fact({[sleep?1,{subject,[#,7]}]@@{simple,[#,9]},[#,8]}, claimed),
fact({card,[#,7],=1}, claimed),
fact({[man?1]@@[#,9],[#,7]}, claimed),
fact({card,[#,10],=1}, assimilated),
fact({[park?1]@@A,[#,10]}, assimilated),
fact({{time,[past]},[#,9]}, assimilated)]
```

Figure 8.13: Examples of temporarily added information.

8.4 Natural Logic Matcher

As mentioned earlier, the engine's PROOF ALGORITHM involves matching decisions (whether the thing to be proven matches an existing fact, a rule consequent, etc.) Such decisions are not obtained by straight unification, but by a natural logic-based matcher. The matcher relies on marked polarities to determine containment relations between pairs. Therefore, given two marked clauses (X and Y), the matcher decides if the latter can be matched to the first based on the (MATCHING ALGORITHM) below.

(MATCHING ALGORITHM)

Given two clauses `X` and `Y`, the matcher can match `Y` to `X` if:

(MA-1) They are unifiable clauses.

(MA-2) They are identical words.

(MA-3) They hold a hyponymy relation:

- (MA-3.1) `hyp(X,Y,_)` if X is a positively marked word.
- (MA-3.2) `hyp(Y,X,_)` if X is a negatively marked word.

(MA-4) They match after deleting a modifier (subsective or intersective only) from:

- (MA-4.1) X if X is a positively marked, or
- (MA-4.2) Y if X is a negatively marked.

(MA-5) They are equal terms (see EQUALITIES).

To demonstrate, consider the inference problem in (8.3) and the IFFs of its p_1 and q in Figure 8.14. From the query, tense and '*John*' are already resolved. Therefore, the engine will attempt to find proof for '*a woman*' that is involved in a '*love*' event. The short version of their proof (omitting trivial matches, when two clauses are unifiable) is illustrated in Figure 8.15 and proceeds as explained in Section 8.2. In contrast to the previous examples, some matching decisions for this proof required more than unification. For instance, the proof of {`[woman? -1]@@[#,3],A`} required the application of rule (MA-4.1) to allow matching it to the fact {`[woman?1,` **`modifier(amod(simple),*(-),[pretty?1])`**`]@@[#,3],[#,1]`} after deleting its subsective modifier.

(8.3)

p_1.	John loves *a pretty woman*$^{\uparrow}$.
q.	Does John love a woman? [Yes]

```
** FACTS **
[fact({([love?1,{dobj,[#,1]},{subject,[#,4]}]@@{simple,[#,3]},[#, 2])},claimed),
 fact({card,[#,1],=1}, claimed),
 fact({[woman?1,modifier(amod(simple),* (-),[pretty?1])]@@[#,3],[#,1]},claimed),
 fact({card,[#,4],=1}, assimilated),
 fact({{name,[John?1]}@@A,[#,4]}, assimilated),
 fact({{time,[now]},[#,3]}, assimilated)]
```

```
**QUERY**
query(the([#,3]::{{time,[now]},[#,3]},
  (({[woman? -1]@@[#,3],A}& {card,A,=1})
    & the(([#, 4]:: ({({name,[den(John)]}@@[#,3],[#, 4])}& {card,[#,4],=1})),
      {(([love?-1,{dobj,A},{subject,[#,4]}]@@ {simple,[#,3]}),B)}))))
```

Figure 8.14: The IFFs of the premise and query of the argument in (8.3).

```
** PROOF **
Trying Horn proof of conjunction c BI-3:
{[woman? -1]@@[#,3],A}&{card,A,=1}
  &{[like? -1,{dobj,A},{subject,[#,4]}]@@{simple,[#,3]},B}
  Trying Horn proof of conjunction c1 BI-3:
  {[woman? -1]@@[#,3],A}&{card,A,=1}
    Looking for a fact that matches c1.1 BI-1:
    Try matching c1.1 to:
    {[woman?1,modifier(amod(simple),* (-),[pretty?1])]@@[#,3],[#,1]},
    Try the matching after deleting a modifier MA-4:
       woman and woman are identical MA-2
       [#,3] and [#,3] are unifiable MA-1
       A and [#,1] are unifiable MA-1
    Match succeeded
    {[woman? -1]@@[#,3],A} matches a claimed fact
    {[woman?1,modifier(amod(simple),* -,[pretty?1])]@@[#,3],[#,1]}
    Proof of c1.1 succeeded
    Looking for a fact that matches c1.2 BI-1:
    ...
    {card,[#,1],=1} matches a claimed fact {card,[#,1],=1}
    Proof of c1.2 succeeded
  Proof of conjunction c1 succeeded
  Looking for a fact that matches c2 BI-1:
    ...
   {[love? -1,{dobj,[#,1]},{subject,[#,4]}]@@{simple,[#,3]},B}
   matches a claimed fact
   {([love?1,{dobj,[#,1]},{subject,[#,4]}]@@{simple,[#,3]},[#, 2])}
  Proof of c2 succeeded
Proof of conjunction c succeeded
Answer: Yes
```

Figure 8.15: CSATCHMO+ proof for the argument (8.3).

8.4.1 Equalities

As seen in Section 7.6.1, equalities have a specific assertable format and follow a certain procedure to get added to the BK. One of the procedural steps involves sorting the arguments of an equality clause. Therefore, to avoid missing out a potential proof because of a wrong order, the process of proving an equality clause {X=Y}@@CTXT, given in (EQUALITY ALGORITHM), starts with sorting its arguments. Then the list of available equalities is searched either for: an eclass whose arguments directly unify with X and Y and are situated at the same time (given by CTXT) (EQ-2.1), or a sequence of eclasses that link X and Y transitively (EQ-2.2). In case neither of these attempts succeeds, the list of equalities is used to de-reference both X and Y to their last values on their chain of equalities (if any), and then re-do (EQ-2.1) and (EQ-2.2). It is possible that X and Y get de-referenced to the same term. In such case, the equality clause is said to be proved by reflexivity (EQ-2.3).

(EQUALITY ALGORITHM)

CSATCHMO+ can prove an equality clause {{X=Y}@@CTXT}, by:

(EQ-1) Ensuring that X and Y are sorted.

(EQ-2) Proving that X and Y are equals at the time CTXT either:

- (EQ-2.1) *Directly:* by finding a matching tuple eclass(X,Y,CTXT,STATUS), or
- (EQ-2.2) *Transitively:* by finding a sequence of tuples: eclass(X,Z1,CTXT,STATUS), eclass(Z1,Z2,CTXT,STATUS)...**and** eclass(Zn,Y,CTXT,STATUS) that prove X and Y are equals transitively, or
- (EQ-2.3) *Reflexively:* by proving that X and Y are the same value (i.e. identical).

(EQ-3) If EQ-2 fails to prove the equality, then X and Y are to be de-referenced, as illustrated in Section 7.6.1, to the last value of their chain of equalities X' and Y' respectively, and then it does EQ-1 and EQ-2 again.

The above algorithm, although it allows answering queries about simple chains of equalities as in (8.4), the intended goal was substitutability in proofs. Thus, consider the example in (8.5) and its relative IFFs in Figure 8.16. After resolving all of the REs in the query, the engine is left with a loving event to prove. As the proof in Figure 8.17 illustrates, the engine proved that event by matching it to the fact {([love?1,{dobj,[#,5]},{subject,[#,1]}]@@{simple,[#,4]}, [#, 2])}. Both loving events were directly matched, except for their subjects. The query subject is the

term [#,7], while for the fact it is [#,1]. Therefore, CSATCHMO+ had to prove that these two terms co-refer to the same object by proving they are equal by reflexivity (they both referred to the same term [#,1] after being de-referenced).

(8.4)

p_1. John is a man.
p_2. The man is a fool.
---
q. Is John a fool? [Yes]

(8.5)

p_1. John is a man who loves Mary.
p_2. The man is a fool.
---
q. Does the fool love Mary? [Yes]

```
** FACTS **
[fact({card,[#,7],=1}, claimed),
fact({[fool?1]@@[#,4],[#,7]}, claimed),
fact({card,[#,1],=1}, claimed),
fact({([love?1,{dobj,[#,5]},{subject,[#,1]}]@@{simple,[#,4]},
[#, 2])},
claimed),
fact({[man?1]@@[#,4],[#,1]}, claimed),
fact({card,[#,6],=1}, assimilated),
fact({{name,[John?1]}@@A,[#,6]}, assimilated),
fact({card,[#,5],=1}, assimilated),
fact({{name,[Mary?1]}@@B,[#,5]}, assimilated),
fact({{time,[now]},[#,4]}, assimilated)]
** EQUALITIES **
[eclass([#, 6], [#, 1], {simple,[#,4]}, claimed),
eclass([#, 7], [#, 1], {simple,[#,4]}, claimed)]
```

```
**QUERY**
query(the([#,4]::{{time,[now]},[#,4]},
  the(([#, 7]:: ({[den(fool)]@@[#,4],[#,7]}& {card,[#,7],=1})),
    the(([#, 5]:: ({({name,[den(Mary)]}@@[#,4],[#, 5])}& {card,[#,5],=1})),
      {(([love?-1,{dobj,[#,5]},{subject,[#,7]}]@@ {simple,[#,4]}),A)})))
```

Figure 8.16: The IFFs of the premise and query of the argument in (8.5).

```
** PROOF **
Trying Horn proof of C :
{(([love?-1,{dobj,[#,5]},{subject,[#,7]}]@@ {simple,[#,4]}),A)}
Looking for a fact that matches C BI-1 :
Try matching C to:
{([love?1,{dobj,[#,5]},{subject,[#,1]}]@@{simple,[#,4]},[#, 2])}
  ...
  Try matching terms MA-5 : [#,7] and [#,1]
    de-ref [#,7] to [#,1] EQ-3
    de-ref [#,1] to [#,1] EQ-3
    [#,7] and [#,1] are equals by reflexivity EQ-2.3
  Match succeeded
  ...
{(([love?-1,{dobj,[#,5]},{subject,[#,7]}]@@ {simple,[#,4]}),A)}
matches a caimed fact
{([love?1,{dobj,[#,5]},{subject,[#,1]}]@@{simple,[#,4]},[#, 2])}
Proof of C succeeded
Answer: Yes
```

Figure 8.17: CSATCHMO+ proof for the argument (8.5).

8.5 Higher-Order Inferences

8.5.1 Defaults

As seen in the engine's (PROOF ALGORITHM) (BI-2), one way of proving something is by matching it to the LHS of a backward rule and prove its RHS. One special kind of backwards rules are Reiter's style defaults (which are of the form: `Y<=(X & consistent(Y))`). The RHS of a default rule contains `consistent(Y)`. The proof of `consistent(Y)` is called (CONSISTENCY CHECK) and it ensures that nothing in the engine's list of available information contradicts the entailed consequence `Y`. That check is achieved by showing that `Y => absurd` cannot be proved.

(CONSISTENCY CHECK)
(CONS-1) `consistent(Y)` is proved, if `Y => absurd` cannot be proved.

To demonstrate, consider the argument in (8.6) and the relevant IFFs in Figure 8.18. As the proof in Figure 8.20 shows, the bit of (q) that needs proving is {`(([fly?-1,` ...} and it matches the LHS of the default rule. Therefore, the proof proceeds with an attempt to prove the RHS of that default rule. The first conjunct of the RHS is proved from the claimed fact `fact({[penguin?1]...})` using the hyponymy relation

between '*penguin*' and '*bird*'. The second conjunct is the consistency check which the engine managed to prove successfully by failing to prove the negation of its clause: `{[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}=>absurd`.

(8.6)

p_1.	Most birds fly.
p_2.	Tweety is a penguin.
q.	Does Tweety fly? [Yes, with defaults]

Now consider adding '*penguins do not fly*' as a third premise to (8.6). As Figure 8.19 shows, that premise extends the list of backward rules. Thus, if engine were to prove the same (q) again, it would fail in proving the consistency check this time due to that rule. Put differently, the new rule allows proving the contradiction: `{[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}=>absurd`, thus, the answer to (q) will be `No`.

```
** FACTS **
[fact({card,[#,9],=1}, claimed),
fact({[penguin?1]@@[#,8],[#,9]}, claimed),
fact({card,[#,11],=1}, assimilated),
fact({{name,[Tweety?1]}@@A,[#,11]}, assimilated),
fact({{time,[now]},[#,8]}, assimilated)]
** BACKWARDS RULES **
[({[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}
  <= (({[bird?0]@@[#,8],A}
      & consistent({[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}))
: claimed))]
** EQUALITIES **
[eclass([#, 11], [#, 9], {simple,[#,8]}, claimed)]
```

```
** QUERY**
query(the([#,8]::{{time,[now]},[#,8]},
  the(([#, 11]:: ({({name,[den(Tweety)]}@@[#,8],[#, 11])}& {card,[#,11],=1})),
    {(([fly?-1, {subject,[#,11]}]@@ {simple,[#,8]}),A)})))
```

Figure 8.18: The IFFs of the premise and query of the argument in (8.6).

```
** FACTS **
[fact({card,[#,9],=1}, claimed),
fact({[penguin?1]@@[#,8],[#,9]}, claimed),
fact({card,[#,11],=1}, assimilated),
fact({{name,[Tweety?1]}@@A,[#,11]}, assimilated),
fact({{time,[now]},[#,8]}, assimilated)]
** BACKWARDS RULES **
(absurd
  <= (({card,A,B}
    & ({[penguin? -1]@@[#,8],A}
      & {[fly? -1,{subject,A}]@@{simple,[#,8]},C}))
      : claimed))
[({[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}
  <= (({[bird?0]@@[#,8],A}
    & consistent({[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}))
: claimed))]
** EQUALITIES **
[eclass([#, 11], [#, 9], {simple,[#,8]}, claimed)]
```

Figure 8.19: The IFFs of Figure 8.18 after adding '*penguins do not fly*'.

8.5.2 Attitude Clauses

As mentioned earlier, the entailment decision about an attitude clause's implicit proposition (`Prop`) is determined by some (ATTITUDES RULES). These rules represent the basic **entailing** patterns, the first four among ($++$, $+-$, $-+$, $--$, $+0$ and -0), that lead to an entailment about either `Prop` or its negation. During a proof, when an attitude clause is unified with one of the (ATTITUDES RULES), the value of `Prop` either gets added to the list of available information or proved. Since skolemization decisions, as discussed in Section 7.5.7, vary depending on whether the context is positive or negative, `Prop`s are usually left in their `raw` form (Section 7.3.5) until it is clear how they are to be used, meaning that, once the proof engine decides whether a `Prop` is to be added to the KB or to be proved, the proof process gets interrupted and the tree normalizer gets asked to resolve `Prop` from a `raw` form to an IFF (RESOLVE RAW).

Consider, for example, proving (q) from (p_1) in (8.7). Given their IFFs in Figure 8.22, it can be seen that the attitude clause of '*manage*' unifies with the forward rule (ATT-1), meaning that its embedded **`raw(...)`** is resolved (RAW-1) and added to the list of available information as a fact {`sleep?1,...`}. Having that fact added makes proving (q) a straightforward application of rule (BI-1).

(8.7)

p_1.	John managed to sleep.
q.	Did John sleep? [Yes]

```
** PROOF **
Trying Horn proof of C :
{[fly? -1,{subject,[#,11]}]@@{simple,[#,8]},A}
Looking for a fact that matches C BI-1 :
Unsuccessful
Looking for a rule whose LHS matches C BI-2 :
Try matching C to the LHS of:
{[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}
 <=({[bird?0]@@[#,8],A}& consistent({[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]})
   ...
   fly and fly are identical MA-2
   [subject,[#,11]] and [subject,[#,11]] are unifiable MA-1
   ...
Match succeeded
Trying Horn proof of the RHS BI-2 :
{[bird?0]@@[#,8],A}
  & consistent{[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}
Trying Horn proof of conjunction RHS BI-3 :
  Looking for a fact that matches RHS.1 BI-1 :
  Try matching RHS.1 to:
  {[penguin?1]@@[#,8],[#,9]}
     penguin is a hyponym of bird MA-3.1
     ...
  Match succeeded
  Proof of RHS.1 succeeded
  Trying Horn proof of consistency RHS.2 CONS-1 :
  consistent{[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}
  Trying Horn proof of:
  {[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}=>absurd
  ...
  About to try conditional proof BI-5 by asserting:
  {[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}
  And trying to prove: absurd
     Looking for a fact that matches absurd BI-1 :
     Unsuccessful
     ...
  Tried every way of getting a Horn proof of:
  {[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}=>absurd
  Consistency check RHS.2 succeeded
Proof of conjunction RHS succeeded
Proof of C succeeded
Answer: Yes, with defaults:
[consistent({[fly?1,{subject,[#,11]}]@@{simple,[#,8]},[#,7]})]
```

Figure 8.20: CSATCHMO+ proof for the argument (8.6).

On the other hand, answering the same query from (p_1) in (8.8)(Figure 8.23) needs a bit more work. As the proof in Figure 8.24 shows, the engine failed to prove C and

```
** PROOF **
Trying Horn proof of c :
{[fly? -1,{subject,[#,11]}]@@{simple,[#,8]},A}
...
  Trying Horn proof of consistency RHS.2 cons-1 :
   consistent{[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}
  Trying Horn proof of :
  {[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}=>absurd
  ...
  About to try conditional proof bi-5 by asserting:
  {[fly?1,{subject,A}]@@{simple,[#,8]},[#,7]}
  And trying to prove: absurd
     Looking for a fact that matches absurd bi-1 :
     Unsuccessful
     Looking for a rule whose LHS matches absurd bi-2 :
     Try matching absurd to the LHS of:
     (absurd
      <= (({card,A,B}
        & ({[penguin? -1]@@[#,8],A}
          & {[fly? -1,{subject,A}]@@{simple,[#,8]},C}))
     Match succeeded
     Trying Horn proof of RHS bi-2 :
     (({card,A,B}
       & ({[penguin? -1]@@[#,8],A}
         & {[fly? -1,{subject,A}]@@{simple,[#,8]},C})
     Trying Horn proof of conjunction RHS bi-3 :
      ...
     Horn proof of conjunction RHS succeeded
  Conditional proof succeeded
  Consistency check RHS.2 failed
Proof of conjunction RHS failed
Proof of c failed
Answer: No
```

Figure 8.21: CSATCHMO+ proof for the argument in (8.6) after adding '*penguins do not fly*'.

attempted to prove its negation c′. Proving the negation proceeded with a search for a rule whose LHS is absurd and its RHS can be proved. One successful match was the rule (ATT-2) whose RHS is a conjunction of two clauses. The first conjunct RHS.1 was successfully proved as it is unified with the fact {(([fail?1,...}. The second conjunct RHS.2, which is a raw form, was also proved after being resolved RAW-2.

(8.8)

p_1.	John failed to sleep.
q.	Did John sleep? [No]

Attitude rules include some variants of the above basic ones. One variant concerns

(ATTITUDES RULES)

(ATT-1) `{[Att_verb?1, {xcomp([_/ 1,_]),Prop},...]@@CTXT,X} => Prop`

(ATT-2) `({[Att_verb?1,{xcomp([_/-1, _]),Prop},...]@@CTXT X}& Prop)=>absurd`

(ATT-3) `(({[Att_verb?-1,{xcomp([_,_/ -1]),Prop},...}]@@CTXT X} =>absurd)&Prop) =>absurd`

(ATT-4) `({[Att_verb?-1,{xcomp([_,_/1]),Prop},...]@@CTXT,X}=>absurd) =>Prop`

(RESOLVE RAW)

(RAW-1) If a `Prop` is to be added as a claim, then turn it into a QFF with `1` as initial value for polarity marking, then to an IFF.

(RAW-2) If a `Prop` is to be proved as a query, then turn it into a QFF with `-1` as initial value for polarity marking, then to an IFF.

```
** FACTS **
[fact({[sleep?1,{subject,[#,9]}]@@{simple,[#,8]},[#,-2]}, claimed),
fact({[male?1]@@C,[#,9]}, claimed),
fact({(([manage?1,
{(xcomp([1/1, -1/-1]),
raw(qq(exists, D, {[den(sleep),arg(subject,[#,9])]@@{simple,[#,8]},D}))))},
 {subject,[#,9]}]@@ {simple,[#,8]}),[#, 7])},claimed),
fact({card,[#,9],=1}, assimilated),
fact({{name,[John?1]}@@E,[#,9]}, assimilated),
fact({{time,[past]},[#,8]}, assimilated)
```

```
**QUERY**
query(the([#,8]::{{time,[past]},[#,8]},
  the(([#, 9]:: ({{name,[den(John)]}@@[#,8],[#,9]}& {card,[#,9],=1})),
    {(([sleep?-1, {subject,[#,9]}]@@ {simple,[#,8]}),A)})))
```

Figure 8.22: The IFFs of the premise and query of the argument in (8.7).

the '*used to*' construction that (when positively asserted) has two consequences – that the proposition in question `Prop` held before now, but it does not hold now (Section 5.8.2). For example, '*used to be a businessman*' means someone **was** a businessman but **is** no longer one. Therefore, when one of these consequences is questioned, the rule

```
** FACTS **
[fact({[male?1]@@C,[#,9]}, claimed),
fact({(([fail?1,{(xcomp([1/-1, -1/1]),
  raw(qq(exists, D, {[den(sleep),arg(subject,[#,9])]@@{simple,[#,8]},D}))))},
{subject,[#,9]}]@@ {simple,[#,8]}),[#, 7])},claimed),
fact({card,[#,9],=1}, assimilated),
fact({{name,[John?1]}@@E,[#,9]}, assimilated),
fact({{time,[past]},[#,8]}, assimilated)]]
```

Figure 8.23: The IFFs of the premise and query of the argument in (8.8).

ensures the right answer by paying attention to its tense. Another variant expresses some relationships between attitude verbs such as '*believing something*' entails '*not doubting it*' and vice versa. It is the existence of verbs like this which have positive and negative consequences that leads us to leave embedded clauses in their raw form inside the IFF of the embedding sentence: '*John used to love Mary*' has the positive consequence that he **did** love her and the negative consequence that he does **not** love her now: these two consequences each require a different treatment of '*John loves Mary*', and hence it is not appropriate to embed a single fixed version of its IFF in the IFF for the whole sentence.

8.6 The Engine's Performance, Soundness and Completeness

CSATCHMO+, as it stands, can answer queries fairly fast (in fractions of a second). However, there are two factors that **could** potentially slow it down. The first is that it could get stuck in a loop. It is pretty easy to block this for the backward chaining part of the engine but more difficult with the forward chaining part, particularly when applying higher-order rules. We deal with it by blocking backward chaining proofs if the current goal subsumes a goal already in the goal stack and simply setting a resource bound for the forward chaining part. The second is that although most of our inferences are Horn, having a matcher as part of the engine makes it very difficult for us to do indexing on the main functor of the head of such clauses. Put differently, if what we are using is straight unification, then proving for instance HUMAN(X) would mean only looking for rules that have HUMAN(_) as their consequent, and that can be done very quickly if we index rules by the functor of the consequent (as is done, for instance, in Prolog). However, in CSATCHMO+, the natural logic matcher explores all possible

```
** PROOF **
Trying Horn proof of c :
{(([sleep?-1, {subject,[#,9]}]@@ {simple,[#,8]}),A)}
...
Tried every way of getting a Horn proof of c

Trying to answer the contrary c/ :
{(([sleep?1, {subject,[#,9]}]@@ {simple,[#,8]}),[#, 11])}=> absurd
...
About to try a conditional proof of c/ BI-5 :
Trying to prove c/ by asserting c1/ :
{(([sleep?1, {subject,[#,9]}]@@ {simple,[#,8]}),[#, 11])}
And trying to prove c2/ : absurd
  Looking for a fact the matches c2/ :
  Unsuccessful
  Looking fo a rule whose LHS matches c2/ :
  Try matching c2/ to the LHS of ATT-2 :
    absurd and absurd are unifiable MA-1
  Matching succeeded
  Try Horn proof of the RHS of ATT-2 :
  Trying Horn proof of conjunction RHS BI-3 :
  {[Att_verb? 1, {xcomp([_/ -1, _]), Prop},...]@@CTXT, X}
   & Prop
    Looking of a fact that matches RHS.1 BI-1 :
    Try matching RHS.1 to:
    {(([fail?1,{(xcomp([1/-1, -1/1]),
      raw(qq(exists, D, {[den(sleep),arg(subject,[#,9])]@@{simple,[#,8]},D})
    Matching succeeded
    ...
    Proof of RHS.1 succeeded
    Trying Horn proof of RHS.2 :
    raw(qq(exists, D, {[den(sleep),arg(subject,[#,9])]@@{simple,[#,8]},D})
    Trying Horn proof of RHS.2 after RAW-2 :
    {[sleep? -1,subject,[#,9]]@@simple,[#,8],D}
    Looking for a fact that matches RHS.2 :
    Try matching RHS.2 to:
    {(([sleep?1, {subject,[#,9]}]@@ {simple,[#,8]}),[#, 11])}
      ...
    Matching succeeded
    {[sleep? -1,subject,[#,9]]@@simple,[#,8],D} matches a temp(refutation) fact
    {(([sleep?1, {subject,[#,9]}]@@ {simple,[#,8]}),[#, 11])}
    Proof of RHS.2 succeeded
  Proof of RHS succeeded
Proof of c/ succeeded
Answer: No
```

Figure 8.24: CSATCHMO+ proof for the argument (8.8)

facts and rules to see if there is any potential (given the set of hyponym relations in the background) for matching e.g. (MAN(_), WOMAN(_), etc.) to HUMAN(X). Again, doing so has not slowed down the inference engine (in its current state) since the size of the available information to be searched is not huge. However, if we were able to get all the background information available to a typical adult speaker of a language in the format we use, there is a significant chance that the engine could become slow.

It is common practice to discuss a proof system in terms of soundness and completeness. Both are properties that express some relationship between a proof theory and a model theory. A proof system is sound if it only proves things that are true in every model, and it is complete if it can prove everything that is true in every model. Having a model theory is key here and in this work we have deliberately not gone for one. In other words, our focus is NLIs, i.e. being able to prove what a human could reasonably conclude given a piece of language (this is why we refer to the engine as Constructive SATCHMO+). Thus, the only test we are interested in is whether we get the correct answers as prescribed by the FraCaS annotators. It is therefore not meaningful to discuss soundness and completeness in the current setting. If, in common with other constructive logics, we do not have a model theory then it is not possible to discuss the relationships between the proof theory and the model theory.

It is worth considering whether the proof algorithm terminates. Given that our framework includes both default inference and higher-order inference, it is in fact clear that without the presence of a resource bound the proof search is potentially non-terminating. First-order logic is semi-decidable (you can guarantee to find a proof within a finite time if there is one, but if there is not then the search may never return). Default logic is not even semi-decidable, since you have to show that using the default rules does **not** lead to a contradiction, i.e. you have to show that a certain kind of proof does not exist. Higher-order logic is even worse, since unification of terms is replaced by proving that one subsumes the other, which in turn means proving that one **entails** the other if they are propositions. In other words, even showing that two terms match is undecidable (since it may involve a default proof). The algorithm **does** terminate, but only because we force it to by imposing resource bounds.

8.7 Summary

In this chapter we have explained how we adapted Ramsay's constructive STACHMO and extended it to CSATCHMO+ (CO5). The extensions were twofold. The first is a

matching algorithm that was based on ideas from natural logic and used in CSATCHMO+ to match two tree pairs instead of unifying them (CO5.1). The second is different kinds of inference rules, including those for dealing with extensions to the basic first-order engine (equalities, defaults, attitude clauses), that have been hand-coded into the engine's list of available information (CO5.2). The chapter has also showed a classification of the available information based on their validity during proofs.

Chapter 9

Evaluation and Discussion

In previous chapters, we learned how an NL premise/query is turned into an IFF and how CSATCHMO+ (given an argument in such forms) would answer a query. n this chapter, we provide an empirical assessment of:

- how well IFFs have managed to capture the meaning of the intended semantic phenomena.
- whether CSATCHMO+ succeeded in reasoning with these forms.
- our NLI system's performance in comparison with related systems' performances.

In Section 9.1 of this chapter we discuss what NLI dataset (among the ones surveyed in Chapter 2) we used to evaluate our NLI system and why. In Section 9.2 we present our system's performance on a chosen set of examples from that dataset. Finally, we compare our performance with the related systems' performances in Section 9.3.

9.1 Choosing the FraCaS test-set for evaluation

As presented in Chapter 2, there exist several sources for inference tasks that could be used for evaluating a NLI system, including: 1) the FraCaS test-set, 2) the RTE challenges, 3) the SNLI and MNLI datasets, and 4) the SICK dataset. These sources vary in their characteristics and are generally constructed with a particular goal in mind (see Table 9.1 for a summary of their main characteristics, merits, and drawbacks).

Dataset Name	Characteristics	Merits	Drawbacks
FraCaS	Consists of 346 NLI problems. Problems are explicitly grouped and annotated for the semantic phenomena they represent. Problems were constructed to bear a resemblance to those of semantic text books.	Covers a wide range of semantic phenomena and supports multi-premises inferences. Solving FraCaS problems requires little background knowledge. Multilinguality.	The test suite is rather small. The problems are quite artificial and sometimes ambiguous. Not representative enough for wide coverage evaluations and comparisons of NLI systems.
RTE	Has eight releases (see Table 2.3 for information about each release). Problems were chosen from real sources and focus on forms of inferences.	Larger than the FraCaS test set. Remedies the unnaturalness of constructed examples.	Still too small for training deep learning approaches. Problems are syntactically complex and involve fairly long sentences. Solving RTE problems presupposes the existence a vast amount of world knowledge.
SICK	Consists of 10K NLI problems. Problems consist of short sentences that were generated from image and video captions. Problems were labelled for entailment and relatedness using crowdsourcing. The dataset is specifically tailored for CDSMs.	Rich in lexical, syntax, and semantic phenomena that CDSMs are expected to account for Includes examples that one would consider as logical inferences [Chatzikyriakidis et al., 2017].	Being specifically tailored of CDSMs; several important phenomena and fine grained details (e.g. tense information) have been normalized away.
SNLI	Consists of 570K pairs of English sentences. Using crowd-sourcing, problems were generated (from image captions) and labelled for entailment.	Considered as the first large-scale data set that supports deep learning.	Coming from one genre (written image captions) , it falls short when it comes to representing phenomena such as: propositional attitudes, modality, and temporal adverbs.
MNLI	Consists of 433k NLI problems. Problems were generated and labelled for entailment using crowdsourcing.	Supports deep learning approaches. Amends SNLI short comings by using ten distinct genres of written and spoken English as source of NLI problems.	

Table 9.1: A summary of the NLI datasets discussed in Chapter 2.

We want to evaluate our system's expressiveness and inferential adequacy with regard to a particular set of semantic phenomena that it was designed for–namely generalized quantifiers, defaults and propositional attitudes. For that, we chose the FraCaS test-set mainly because none of the other datasets (listed in Table 9.1) has a clear classification or annotation of what kind of phenomena each group of NLI problems represents [Pavlick, 2017]. Therefore, evaluations based on such data-sets usually focus on the single accuracy metric [White et al., 2017] that, in the words of Poliak [2020], only shows:

> ...how well a model can recognize whether one sentence likely follows from another, but it does not illuminate how well NLP models capture different semantic phenomena that are important for general NLU...

For instance, consider the NLI systems discussed in Section 4.3 and their performances on some datasets given in Table 9.2. It can be noticed that the system of A17 has a better handle on generalized quantifiers (section 1 of the FraCaS test-set – see Table 2.1) with 95% accuracy than the order-based approach of H19a and a better handle on attitudes (section 9 of the FraCaS) than the NatLog system of MM08. However, based on the performances on the test part of the SICK data-set, all that can be said is that the natural tableau based approach of A15 is better than the order-based approach of H19b on doing NLIs, without any breakdown into the systems' performance on different kinds of problem. It should be noted that all the systems that use FraCaS as test-set use the whole set for testing, rather than splitting it into development and test sets. The key here is that each example aims to capture some specific phenomenon, with very little crossover between cases. The fact that a system can handle questions that arise from '*a few great tenors sing popular music.*', for instance, does not tell you whether it will be able to handle ones that arise from '*few great tenors are poor*'. Splitting the test-set into examples to be used for development and ones used for testing would simply remove half the problems, and hence would make it considerably less taxing. The real question is how many **kinds** of problem can a system solve.
In addition to the above, the FraCaS test-set has some other advantages for our purposes:

- It is the most widely used test-set that was designed to model fine-grained semantic phenomena (e.g. the differences between the quantifiers '*few*' and '*a few*') and to support multi-premise inferences (see below). In particular, it has been used to evaluate the NLI systems described in Section 4.3, and hence gives us the best

NLI systems \ Dataset	FraCaS						RTE-3	SICK
	1	2	5	6	9	Total		
MM08	97.7%	75%	80%	81.3%	88.9%	87%	59%	–
A15	–	–	–	–	–	–	–	82.1%
A17	95%	73%	77%	–	92%	86.6%	–	–
H19a	88%	–	–	–	–	88%	–	–
H19b	–	–	–	–	–	–	–	77.19

Table 9.2: NLI related systems' (reviewed in Section 4.3) performances on some of the discussed (Section 2.2) NLI datasets. The systems are: MM08: MacCartney and Manning [2008], A15: Abzianidze [2015], A17:Abzianidze [2017a], H19a: Hu et al. [2019a], and H19b: Hu et al. [2019b].

yardstick for the performance of our system on the phenomena we are interested in.

- Its inference tasks were constructed deliberately to minimise the amount of background knowledge required for their solution. The system described here could easily be extended to accommodate background knowledge beyond the relations encoded in WordNet, but that is not the primary focus of our work and hence we have chosen not to use a test-set such as the RTE challenges for evaluating it.

- The sentences in this test-set are fairly short, which helps eliminate factors such as parsing errors. There are sentences with fairly complex structures, such as '*every individual who has the right to live in Europe can travel freely within Europe.*', but almost all the examples are well-formed English sentences which are at least potentially parseable with a grammar-based parser – of the 87 sentences in the sections of the FraCaS that we are interested in, we are able to successfully parse 85. The work described here is concerned with carrying out inference on parsed sentences, not with the development of a parser, and hence a test-set that poses difficult semantic problems on the basis of parseable material is appropriate for evaluating our work.

9.2 Experiments and Results

We evaluated our inference system against sections 1 and 9 of the FraCaS examples (listed in Appendix B.1) using MacCartney and Manning [2007]'s machine readable version[1]. On that version the answers are either *Yes*, *No* or *Unknown*. The number

[1] Available in http://www-nlp.stanford.edu/~wcmac/downloads/FraCaS.xml

of problems used for testing from section 1 are 74 (6 problems are usually excluded because they lack a well-defined answer) and the whole 13 from section 9. For each problem, we compare our answer to the gold one in Appendix C.1. Figure 9.1 classifies our answers against the gold answers for section 1 on the left table, and for section 9 on the right table. The calculation of precision and recall scores is slightly confusing because `Unknown` is sometimes the system's view of the question (when it has failed to prove either the queried proposition or its negation) and is sometimes returned simply because it failed to parse the question or one of the premises. In the former case it may be the right answer or it may be wrong, and hence should be included in the precision, in the latter it counts as not returning an answer at all and should be included in the recall.

Gold \ Our	Yes	No	Unknown
Yes	34	0	3 (0)[a]
No	0	5	0
Unknown	0	0	32

Section 1: generalised quantifiers

[a]There are no cases of Unknown arising in the section from a failure by the parser.

Gold \ Our	Yes	No	Unknown
Yes	5	0	1 (2)[a]
No	0	1	0
Unknown	0	0	4

Section 9: propositional attitudes

[a]Two of the sentences in this section were simply too difficult for our parser to handle.

Figure 9.1: Confusion matrix for CSATCHMO+ on section 1 (left) and Section 9 (right) of the FraCaS test-set.

Section	# problems	Accuracy %	Precision %	Recall %
1. Generalized Quantifiers	74	96	96	96
9. Attitudes	13	77	91	83
Total	87	93	95	94

Table 9.3: Performance of CSATCHMO+ on section 1 and 9 of the FraCaS examples

There were a small number of cases where we believe that the answers provided in MacCartney and Manning's version of FraCaS, discussed below ,are incorrect. In these cases we took what we believe to be the right answer rather than MacCartney and Manning's as our target, but in such cases we did investigate further related premise-question sets.

- For (FraCaS-010) the gold answer is (Yes) and we believe that it should be (Unknown)

FraCaS-010
Gold: Yes **Ours:** Unknown
p_1: Most great tenors are Italian

---

q: Are there great tenors who are Italian?

we believe that (as discussed in Section 5.2) that certain GQs, such as universal quantifiers (e.g. '*all*' and '*every*'), do not assume existential import , and so the entailment of a sentence such as '*There is/are X*' is only possible if there is other premises asserting the existence of some X. That concept applies for the quantifier '*most*', and thus we believe the answer should have been (Unknown). We did include versions of these examples where an additional sentence that does have existential import (e.g. '*There are some great tenors*') in order to obtain '*Yes*' as the answer.

- For (FraCaS-015), the gold answer is (Yes) and we believe for this one it should have been (Unknown) as well.

FraCaS-015
Gold: Yes **Ours:** Unknown
p_1: At least three tenors will take part in the concert.

---

q: Are there tenors who will take part in the concert?

In other words, if we are really careful about times, then actually the question should be '*will there be tenors who will take part in the concert?*', since we do not know that the ones who will take part have yet been born/identified as tenors. Consider as a parallel example '*at least three cakes will be made for the wedding*' $\models$ '*will there be cakes.*' but not '*there are three cakes.*'.

Again we tested a version of this example for which we believe the answer should be Yes, namely one where the question was '*Will there be tenors who will take part in the concert?*' and obtained the answer Yes. There are a number of places where the FraCaS answers lack precision with respect to tense and aspect marking, but this and (FraCaS-334) are the only cases where it makes a significant difference.

- For (FraCaS-345) and (FraCaS-346), the gold answer is (Yes) and we obtained (Unknown). Both examples (below) involved with a conjunction or a disjunction between two sentences. Cases like (FraCaS-345) involving conjunctions or disjunctions of sentences are comparatively easy to handle. There are, however,

cases where the linked items are not sentences: these are much harder, e.g. '*John ate a peach or a pear*', where care has to be taken not produce an interpretation which involves saying there is a peach and a pear and John ate one or other of these two. We have left handling coordination for future work, and hence have not dealt with cases like (FraCaS-345) and (FraCaS-346).

FraCaS-345
Gold: Yes **Ours:** Unknown
p_1: Smith saw Jones sign the contract and his secretary make a copy.

---

q: Did Smith see Jones sign the contract?

FraCaS-346
Gold: Yes **Ours:** Unknown
p_1: Smith saw Jones sign the contract or cross out the crucial clause.

---

q: Did Smith either see Jones sign the contract or see Jones cross out the crucial clause?

- (FraCaS-334) is also worth a closer look:

  FraCaS-334
  Gold: Yes **Ours:** Unknown
  p_1: Smith knew that ITEL had won the contract in 1992.

  ---

  q: did ITEL win the contract in 1992?

  The issue here is that the embedded proposition in the premise '*ITEL had won the contract in 1992.*' has different tense/aspect marking from the query '*did ITEL win the contract in 1992?*'. Assuming that '*in 1992*' in the premise attaches to '*won the contract*' and not to '*knew that ITEL had won the contract*', we still have to be prepared to infer that having done it in 1992 entails doing it in 1992. This can be done either by being sloppy and saying that all past forms are equivalent, or by explicitly having a rule for relating perfect and simple forms. We have not included a complete treatment of the relationships between various tense/aspect combinations: the rule below shows the general form that rules for such a treatment would have.

```
{{time,[past | TIMES]},T0}
&{(P@@{perfect,T0}, X)}
```

```
& {{time, TIMES}, T1}
=> {(P@@{simple,T1}, X)}
```

If P was perfect at some time then it was simple at any later time.

- (FraCaS-343) also raises an interesting issue:

  FraCaS-343
  Gold: Yes **Ours:** Yes
  p_1: Smith saw Jones sign the contract.
  p_2: Jones is the chairman of ITEL.
  ---
  q: did Smith see the chairman of ITEL sign the contract?

  The problem this time is that we need to match the subtrees for '*Jones sign the contract*' from the first premise and '*the chairman of ITEL sign the contract*' from the query. This can only be done if we can match '*Jones*' and '*the chairman of ITEL*', which we clearly **cannot** do unless we can make use of the second premise. We defined matching in Section 8.4 by saying that X matches Y if X is a hyponym of Y. Recalling that hyponymy is just a specific form of entailment, we can extend this to say that X matches Y if X entails Y. Given the second premise of (FraCaS-343), we can show that '*Jones sign the contract*' does entail '*the chairman of ITEL sign the contract*', and hence we can match the corresponding subtrees.

  This extension of the matching algorithm only comes into play when matching the subtrees for embedded clauses, and hence has little effect on the performance of the inference engine on the majority of the FraCaS examples, but it does make it possible to handle some otherwise intractable problems.

9.3 Comparison and Discussion

In this section, the performance of CSATCHMO+ is compared to the related systems (Section 4.3) which have been tested on the FraCaS test-set. The comparison is given in Table 9.4. The results have been split into systems that can do single-premise problems and others that can do single and multiple premise problems for clarity.

As illustrated in the table, from section 1 our inference system has achieved a comparable result to the other systems when evaluated on single-premises only and better results when multi-premise problems are included. Nevertheless, the FraCaS

Section	# problems (Single/All)	Single (Acc %)			All (Acc %)		
		MM08	A17	Ours	A17	H19	Ours
1. Generalized Quantifiers	(44/74)	98	93	96	95	88	96
9. Attitudes	(11/13)	89	100	96	92	-	77

Table 9.4: A comparison between the accuracy of our system (on section 1 and 9 of the FraCaS) and the systems discussed in Section 4.3. MM08: MacCartney and Manning [2008], A17: Abzianidze [2017b] and H19: Hu et al. [2019a].

data is fairly small and its problems are usually seen during systems' development [Abzianidze, 2017b]. Therefore, "the comparison should be understood in terms of an expressive power of a system and the underlying theory" Abzianidze [2017b], rather than just by counting examples. With that being said, we believe that some semantic phenomenon were poorly represented in the FraCaS test-set, particularly including attitudes (Appendix B.1.2) (see Chatzikyriakidis et al. [2017]'s discussion and proposal on this matter). Attitudes, as explained in Section 5.8.2, have a number of important features that ought to be considered in order to make the right inferences, including: their entailment pattern, tense-agreement and their relation with other attitudes. For example, looking into section 9 of the FraCaS data set, it can be noticed that the examples include one implicative verb '*manage*', one factive '*know*' and one non-factive '*believe*', all in positive contexts only. Therefore, to further investigate our system's ability represent such constructions and reason with them, we have extended the test on attitudes into a number of further cases that were mainly constructed based on Karttunen's [1971; 2012; 2015b] examples for implicatives and factives, and some other variants from the ones in the FraCaS itself (all listed in Appendix B.2). We tested our system and Abzianidze's LangPro[2] on this set of examples. As the results in Appendix C.2 show, CSATCHMO+ was able to answer all of these examples. LangPro on the other hand was able to answer a number of these examples, but did not deal with examples where tense information is important (unsurprisingly given that Abzianidze intentionally skip tense information), and others with verbs that (although they are examples of attitudes) are not explicitly mentioned in Abzianidze's thesis . However, there are some straightforward applications of the basic entailment patterns ($++$, $+-$, $-+$, $--$) for which Abzianidze has designed four specific tableau rules that LangPro could not answer correctly, For example:

- (FraCaS-k23) which is a simple variant of (FraCaS-336) where '*managed*' was

[2] https://naturallogic.pro/LangPro/

replaced with '*failed*' and (FraCaS-k09) is another example of '*failed*' but in a negative context.

FraCaS-k23
Gold: No **LangPro:** Unknown
p_1: ITEL **failed** to win the contract in 1992.
---
q: ITEL won the contract in 1992.

FraCaS-k09
Gold: Yes **LangPro:** Unknown
p_1: John **did not fail** to eat a peach.
---
q: John ate a peach.

- (FraCaS-k25b) is about natural logic containment relation between attitudes' complements.

FraCaS-k00
Gold: Yes **LangPro:** Unknown
p_1: John believes **a fat old man loves** Mary.
---
q: John believes **a human likes** Mary.

9.4 Summary

Using the FraCaS test set, we have shown that our system produces comparable performance to that of other systems addressing the same range of problems, in particular complex quantificational issues, and that by adapting a standard inference engine to use natural logic matching on trees rather than the unification of logical forms we can outperform all the competing systems on generalized quantifiers examples. The FraCaS coverage of propositional attitudes is somewhat limited, so we added a collection of examples from work by Karttunen. These are not extracted from freely occurring texts, but nor were they constructed by us for the purpose of testing a system developed by us: they are accepted in the literature as exemplifying a range of challenging issues, and hence our ability to solve them supports our claim that our system outperforms a number of others on a range of significant phenomena.

Chapter 10

Conclusion and Future Work

In this thesis, we have investigated the ability to implement an automated NLI system that could reason about deep semantic phenomena without the need for translation into some formal language such as FOL. To achieve that, we have built an inference system with the following aims and questions in mind:

Aims:

A1. Investigate the use of syntactical analysis of NL texts (dependency trees in particular) as a basis for reasoning about NL.

A2. Re-visit a class of inference problems (exemplified by parts of the Fracas data-set and some others, and centred around certain deep semantic phenomena including defaults, quantifiers, and propositional attitudes) that cannot be handled by simple subsumption relations over trees , due to the lack of mechanisms for chaining over multi-premises, nor by translating into logical expressions (which have been proven difficult) **alone**, and attempt to solve them by designing and implementing a NLI system that uses a combination of a tree-matching algorithm and theorem proving.

Questions:

Q1. Can we use representations that are close to trees obtained by standard syntactic analysis as the MRs for complex issues in semantics?

Q2. Can the target inference problems (in A2) be tackled by adapting existing inference techniques, namely natural deduction rules and natural logic containment relations, to operate directly on those representations?

To answer these questions, we have designed and implemented a NLI system that consists of:

S1. A **parser** that takes a NL utterance and constructs its dependency tree.

S2. A **tree normalizer** that takes a dependency tree and turns it, by applying a series of transformational stages, into a form (inference friendly forms (IFFs)) that the theorem prover can work with.

S3. A **theorem prover** that, given a query (in its IFF), attempts to prove it from any previously mentioned premise(s) (within the current discourse) and any available information in the repository, and conclude its attempt by an answer.

S4. A **repository of information** to be used as background knowledge.

To arrive at the final system above, we have:

- surveyed (in Chapter 4) the literature for natural logic-based NLIs and identified some of their shortcomings (CO1).
- searched the literature (in Chapter 5) to gain a better understanding of how certain semantic phenomena (including defaults, quantifiers, propositional attitudes) have been handled, and showed what representational aspects we have considered to capture their semantics in our MRs, keeping in mind the need: 1) to reason about examples of such phenomena; and 2) bridge the found gaps in the existing NLI systems. (CO2)
- used (in Chapter 6) a dependency grammar to turn NL text into DTs, and ensured that DTs were labelled with the necessary semantic features (CO2.1).
- designed (in Chapter 7) a set of transformation rules, based on what has been learned from the literature on the meaning of the intended semantic phenomena and the demands of the inference engine, then applied them on labelled DTs to obtain their IFFs (CO2.2).
- adapted (in Chapter 8) Ramsay's (2001) constructive SATCHMO to work with normalized trees instead of logical formulas, incorporating two extensions. The first was a matching algorithm that we designed based on ideas from natural logic (CO3), and used instead of straight unification (CO5.2). The second was a number of inference rules that were designed and hand-coded (into the engine repository of information) to handle certain constructions (CO5.1).

- run our inference engine (in Chapter 9) against sections 1 and 9 from the FraCas test-set and compared the engine's answers with the gold standard answers.

Our system achieved 93 % overall accuracy on the intended sections. The problems that the engine failed to answer were either because the parser failed to get their trees or because they involved constructions that were out of scope. With these findings, we believe the constructed IFFs have sufficiently captured the meaning of the intended semantic phenomena and that CSATCHMO+ succeeded in reasoning with these forms.

10.1 Future Work

For future work, we have a number of ideas that could further extend the system presented in this thesis. These ideas can be split into two groups. The first are extensions that we believe are fairly straightforward with more effort. The second is about ideas that might be tempting in principle but might be hard to apply and hence need investigation. On the side of doable things, we want to try our inference system on more NLI problems. In particular, the NLI problems:

1. that constitute other sections (except ellipses) of the FraCaS test-set. We believe that there are a substantial number of semantic phenomena that we have dealt with in parsing—such as bare plurals, anaphora, the standard use and anaphoric dimension of tenses, and non-affirmative adjectives (fake, former, etc.)—and we have covered (to some degree) in normalization. However, we have not gone as carefully through these problems to ensure that their representations have captured the intended semantic phenomena, and we have not tested whether CSATCHMO+ can get their correct answers.

2. of the SICK dataset. Although SICK problems were not labelled for semantic phenomena, it would be interesting to test how well our inference system could handle more NLIs especially knowing that the SICK problems are split into an unseen part for testing and another for developing. In addition, FraCaS and SICK problems share some similarities despite being constructed for different goals. That is, the problems of both datasets consist of short sentences, were manually generated, resemble problems of logical inferences, and require comparatively little world knowledge.

On the other side,

1. It would be really interesting if we could run our system on freely occurring long NL sentences. However, as discussed in Section 6.1, the work described in this thesis relies upon having accurate parse trees that are labelled with the necessary semantic information, and no existing parser we are aware of is accurate or informative enough to do the job. For instance, statistically trained parsers have particular difficulty with sentences containing complex determiners. As noted in Section 6.1, SDP provides completely different analyses to the NPs in '*few great tenors are poor.*' and '*most great tenors are rich*'. Similarly, both SDP and MALTParser assign different structures to the NPs in '*all great tenors are rich*' and '*most great tenors are rich*', and MALTParser assigns different structures to the NPs in '*at least ten commissioners spend time at home.*' and '*do at least ten commissioners spend time at home?*'. The problem seems likely to be that such constructions are comparatively rare, giving the parser little to learn from, and that words like '*most*' and '*few*' tend to have multiple interpretations, making it hard to assign them the right POS tag even before the parser takes over. Given that the issues we are interested in relate very largely to complex determiners of this kind, the output of such parsers is not reliable enough for them to be used for this kind of task. Thus, we have to either use a grammar-based parser with all their known faults or wait for statistical ones to get substantially better on difficult examples.

2. the inference engine could benefit from having more knowledge. As shown in Section 8.3.1, we can extract some useful lexical relations from WordNet quite easily. Nonetheless, extracting large-scale encyclopaedia kind of information will not be easy, and even if we could obtain such information, we would run into two more obstacles:

    - This information has to be encoded into the engine's repository of information in the same format as IFFs.
    - Having them all encoded could potentially (as discussed in Section 8.6) slow down the engine and hence could require finding a way to optimize its performance.

Bibliography

Barbara Abbott. Definite and indefinite. Encyclopedia of language and linguistics, 3 (392):99, 2006.

Barbara Abbott and Bart Geurts. Definiteness and proper names: Some bad news for the description theory. author's reply. Journal of semantics (Nijmegen), 19(2): 191–207, 2002.

Nabil Abdullah and Richard A Frost. Adjectives: A uniform semantic approach. In Conference of the Canadian Society for Computational Studies of Intelligence, pages 330–341. Springer, 2005.

Lasha Abzianidze. Towards a wide-coverage tableau method for natural logic. In JSAI International Symposium on Artificial Intelligence, pages 66–82. Springer, 2014.

Lasha Abzianidze. A tableau prover for natural logic and language. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 2492–2502, 2015.

Lasha Abzianidze. Natural solution to fracas entailment problems. In Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics, pages 64–74, 2016.

Lasha Abzianidze. Langpro: Natural language theorem prover. arXiv preprint arXiv:1708.09417, 2017a.

Lasha Abzianidze. A natural proof system for natural language. Tilburg University, 2017b.

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

Hiyan Alshawi. The core language engine. MIT press, 1992.

Hiyan Alshawi and Jan van Eijck. Logical forms in the core language engine. In 27th Annual Meeting of the Association for Computational Linguistics, pages 25–32, 1989.

Maxime Amblard, Clément Beysson, Philippe de Groote, Bruno Guillaume, and Sylvain Pogodalla. A french version of the fracas test suite. In LREC 2020-Language Resources and Evaluation Conference, page 9, 2020.

Gabor Angeli and Christopher D Manning. Naturalli: Natural logic inference for common sense reasoning. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 534–545, 2014.

Grigoris Antoniou. A tutorial on default logics. ACM Computing Surveys (CSUR), 31(4):337–359, 1999.

N Arthur. Time and Modality. Clarendon Press, Oxford, 1957.

Emmon Bach. The algebra of events. Linguistics and philosophy, 9(1):5–16, 1986.

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second pascal recognising textual entailment challenge. In Proceedings of the second PASCAL challenges workshop on recognising textual entailment, volume 6, pages 6–4, 2006.

Avron Barr. Natural language understanding. AI Magazine, 1(1):5, 1980.

Jon Barwise and Robin Cooper. Generalized quantifiers and natural language. In Philosophy, language, and artificial intelligence, pages 241–301. Springer, 1981.

Johan van Benthem et al. A brief history of natural logic. College Publications, 2008.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. The fifth pascal recognizing textual entailment challenge. Proceedings of TAC, 9:14–24, 2009.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. The sixth pascal recognizing textual entailment challenge. Proceedings of TAC, 2010.

Luisa Bentivogli, Peter Clark, Ido Dagan, Hoa Dang, and Danilo Giampiccolo. The seventh pascal recognizing textual entailment challenge. Proceedings of TAC, 2011, 2011.

Luisa Bentivogli, Ido Dagan, and Bernardo Magnini. The recognizing textual entailment challenges: Datasets and methodologies. In Handbook of Linguistic Annotation, pages 1119–1147. Springer, 2017.

Timothée Bernard. Negation in event semantics with actual and nonactual events. Proceedings of ConSOLE XXVI, 1:17, 2018.

Raffaella Bernardi. Monotonic reasoning from a proof-theoretic perspective. In Proceedings of Formal Grammar, pages 13–24, 1999.

Jean-Philippe Bernardy and Stergios Chatzikyriakidis. A type-theoretical system for the fracas test suite: Grammatical framework meets coq. In IWCS 2017-12th International Conference on Computational Semantics-Long papers, 2017.

Jean-Philippe Bernardy and Stergios Chatzikyriakidis. What kind of natural language inference are nlp systems learning: Is this enough? In ICAART (2), pages 919–931, 2019.

Patrick Blackburn. Tense, temporal reference, and tense logic. Journal of Semantics, 11(1-2):83–101, 1994.

Patrick Blackburn and Johan Bos. Representation and inference for natural language. A first course in computational semantics. CSLI, 2005.

Alexander Bochman. Nonmonotonic reasoning. In Handbook of the History of Logic, volume 8, pages 557–632. Elsevier, 2007.

Rens Bod. The Data-Oriented Parsing Approach: Theory and Application, pages 307–348. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

Johan Bos. Predicate logic unplugged. In Proceedings 10th Amsterdam Colloquium, pages 133–143. Citeseer, 1996.

Johan Bos. Let's not argue about semantics. In Proceedings of the International Conference on Language Resources and Evaluation, pages 2835–2840. Citeseer, 2008.

Johan Bos. A survey of computational semantics: Representation, inference and knowledge in wide-coverage text understanding. Language and Linguistics Compass, 5(6):336–366, 2011.

Johan Bos and Katja Markert. Recognising textual entailment with logical inference. In Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, pages 628–635, 2005.

Johan Bos and Katja Markert. When logical inference helps determining textual entailment (and when it doesnt). In Proceedings of the Second PASCAL RTE Challenge, page 26, 2006.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. arXiv preprint arXiv:1508.05326, 2015.

Gerhard Brewka. Default Reasoning, pages 915–917. Springer US, Boston, MA, 2012. ISBN 978-1-4419-1428-6. doi: 10.1007/978-1-4419-1428-6_634. URL https://doi.org/10.1007/978-1-4419-1428-6_634.

Gerhard Brewka, Jürgen Dix, and Kurt Konolige. Nonmonotonic reasoning: an overview, volume 73. CSLI publications Stanford, 1997.

Gerhard Brewka, Ilkka Niemelä, and Mirosław Truszczyński. Nonmonotonic reasoning. Foundations of Artificial Intelligence, 3:239–284, 2008.

Daniel Büring. Pronouns, pages 971 – 996. De Gruyter Mouton, 2011.

Cambridge Online Dictionary. Minutes, 2021. URL https://dictionary.cambridge.org/dictionary/english/minutes. [Online; accessed 11-August-2021].

Bob Carpenter. Type-logical semantics. MIT press, 1997.

Nathanael Chambers, Daniel Cer, Trond Grenager, David Hall, Chloe Kiddon, Bill MacCartney, Marie-Catherine De Marneffe, Daniel Ramage, Eric Yeh, and Christopher D Manning. Learning alignments and leveraging natural logic. In Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing, pages 165–170, 2007.

Lucas Champollion. Quantification and negation in event semantics. The Baltic International Yearbook of Cognition, Logic and Communication, 6:3, 2010.

Lucas Champollion. Integrating montague semantics and event semantics. ESSLLI lecture notes, 2014.

Stergios Chatzikyriakidis, Robin Cooper, Simon Dobnik, and Staffan Larsson. An overview of natural language inference data collection: The way forward? In Proceedings of the Computing Natural Language Inference Workshop, 2017.

Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 740–750, 2014.

N Chomsky. Conditions on rules of grammar. essays on form and interpretation, 163-210, 1976.

Noam Chomsky. Knowledge of language: Its nature, origin, and use. Greenwood Publishing Group, 1986.

Noam Chomsky, Roderick A Jacobs, and Peter S Rosenbaum. Remarks on nominalization. 1970, 184:221, 1970.

Paul Christophersen. The articles: A study of their theory and use in english. 1939.

Ariel Cohen and Nomi Erteschik-Shir. Topic, focus, and the interpretation of bare plurals. Natural Language Semantics, 10(2):125–165, 2002.

Bernard Comrie et al. Tense, volume 17. Cambridge university press, 1985.

Robin Cooper. Quantification and syntactic theory. Synthese Language Library, 1983.

Robin Cooper, Richard Crouch, Jan van Eijck, Chris Fox, Josef van Genabith, Jan Jaspers, Hans Kamp, Manfred Pinkal, Massimo Poesio, Stephen Pulman, et al. Fracas: A framework for computational semantics. Deliverable, 8:62–051, 1994.

James R Curran, Stephen Clark, and Johan Bos. Linguistically motivated large-scale nlp with c&c and boxer. In Proceedings of the 45th annual meeting of the Association for Computational Linguistics Companion volume proceedings of the demo and poster sessions, pages 33–36, 2007.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment, pages 177–190. Springer, 2005.

Marcello D'Agostino and Marco Mondadori. The taming of the cut. classical refutations with analytic cut. Journal of Logic and Computation, 4(3):285–319, 1994.

Mary Dalrymple. The interpretation of tense and aspect in english. In Proceedings of the 26th annual meeting on Association for Computational Linguistics, pages 68–74. Association for Computational Linguistics, 1988.

Donald Davidson. The logical form of action sentences. In The logic of decision and action. Pittsburgh University Press, 1967.

Andrea DeCapua. Time, tense, and aspect of verbs. In Grammar for Teachers, pages 165–209. Springer, 2008.

Renaat Declerck. From reichenbach (1947) to comrie (1985) and beyond: Towards a theory of tense. Lingua, 70(4):305–364, 1986.

Francesco M Donini, Maurizio Lenzerini, Daniele Nardi, Fiora Pirri, and Marco Schaerf. Nonmonotonic reasoning. Artificial Intelligence Review, 4(3):163–210, 1990.

Myroslava O Dzikovska, Rodney D Nielsen, Chris Brew, Claudia Leacock, Danilo Giampiccolo, Luisa Bentivogli, Peter Clark, Ido Dagan, and Hoa T Dang. Semeval-2013 task 7: The joint student response analysis and 8th recognizing textual entailment challenge. Technical report, DTIC Document, 2013.

Mürvet Enç. Anchoring conditions for tense. Linguistic inquiry, pages 633–657, 1987.

Christiane Fellbaum. Wordnet: An electronic lexical database and some of its applications. MIT press Cambridge, 1998.

Christiane Fellbaum. Wordnet. In Theory and applications of ontology: computer applications, pages 231–243. Springer, 2010.

Hana Filip. 48. aspectual class and aktionsart. Semantics, 2011.

Melvin Fitting. First-order logic and automated theorem proving. Springer-Verlag, 1990.

Yaroslav Fyodorov, Yoad Winter, and Nissim Francez. Order-based inference in natural logic. Logic Journal of the IGPL, 11(4):385–416, 2003.

LTF Gamut. Logic, language, and meaning, volume 2. University of Chicago Press, 1991.

Gerhard Gentzen. Untersuchungen über das logische schließen. i. Mathematische zeitschrift, 39(1):176–210, 1934.

Bart Geurts. Good news about the description theory of names. Journal of semantics, 14(4):319–348, 1997.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third pascal recognizing textual entailment challenge. In Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing, pages 1–9. Association for Computational Linguistics, 2007.

Danilo Giampiccolo, Hoa Trang Dang, Bernardo Magnini, Ido Dagan, Elena Cabrio, and Bill Dolan. The fourth pascal recognizing textual entailment challenge. TAC 2008 Proceedings, 2008.

Jean-Yves Girard. Linear logic. Theoretical computer science, 50(1):1–101, 1987.

Jean-Yves Girard. Linear logic: its syntax and semantics. London Mathematical Society Lecture Note Series, pages 1–42, 1995.

Jean-Yves Girard. Light linear logic. Information and Computation, 143(2):175–204, 1998.

Oren Glickman, Ido Dagan, and Moshe Koppel. Web based probabilistic textual entailment. In Proceedings of the 1st Pascal Challenge Workshop, pages 33–36, 2005.

Herbert P Grice. Logic and conversation. In Speech acts, pages 41–58. Brill, 1975.

D Grune and CJH Jacobs. Parsing techniques-a practical guide, 2008.

Susanne Hackmack. Reichenbach's theory of tense and it's application to english, 2015.

Ilan Hazout. The syntax of existential constructions. Linguistic Inquiry, 35(3):393–430, 2004.

Irene Heim. The semantics of definite and indefinite noun phrases. 1982.

Irene Heim. File change semantics and the familiarity theory of definiteness. Semantics Critical Concepts in Linguistics, pages 108–135, 1983.

Elena Herburger. Negation, volume 2, pages 1641 – 1660. De Gruyter Mouton, 2011.

Caroline Heycock and Anthony Kroch. Inversion and equation in copular sentences. ZAS Papers in linguistics, 10:71–87, 1998.

James Higginbotham. On semantics. Linguistic inquiry, 16(4):547–593, 1985.

James Higginbotham. On events in linguistic semantics. Speaking of events, 49:80, 2000.

Laurence R Horn. Implicature. Encyclopedia of Cognitive Science, 2006.

Hai Hu and Larry Moss. Polarity computations in flexible categorial grammar. In Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, pages 124–129, 2018.

Hai Hu, Thomas Icard, and Larry Moss. Automated reasoning from polarized parse trees. In Proceedings of the Fifth Workshop on Natural Language and Computer Science, 2018.

Hai Hu, Qi Chen, and Larry Moss. Natural language inference with monotonicity. In Proceedings of the 13th International Conference on Computational Semantics-Short Papers, pages 8–15, 2019a.

Hai Hu, Qi Chen, Kyle Richardson, Atreyee Mukherjee, Lawrence S Moss, and Sandra Kübler. Monalog: a lightweight system for natural language inference based on monotonicity. arXiv preprint arXiv:1910.08772, 2019b.

Thomas Icard III and Lawrence Moss. Recent progress in monotonicity. LiLT (Linguistic Issues in Language Technology), 9, 2014.

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

Andrzej Indrzejczak. Natural Deduction, Hybrid Systems and Modal Logics, volume 30. Springer Science & Business Media, 2010.

Lucja Iwańska. Logical reasoning in natural language: It is all about knowledge. Minds and Machines, 3(4):475–510, 1993.

Ray Jackendoff. X syntax: A study of phrase structure. Linguistic Inquiry Monographs Cambridge, Mass, (2):1–249, 1977.

Stanisław Jaśkowski. On the rules of suppositions in formal logic. Studia Logica, (1): 5–32, 1934.

Valentin Jijkoun, Maarten de Rijke, et al. Recognizing textual entailment using lexical similarity. In Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment, pages 73–76. Citeseer, 2005.

Lauri Karttunen. Implicative verbs. Language, pages 340–358, 1971.

Lauri Karttunen. Simple and phrasal implicatives. In * SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012), pages 124–131, 2012.

Lauri Karttunen. From natural logic to natural reasoning. In CICLing, 2015a.

Lauri Karttunen. From natural logic to natural reasoning. In International Conference on Intelligent Text Processing and Computational Linguistics, pages 295–309. Springer, 2015b.

Edward L Keenan and Dag Westerståhl. Generalized quantifiers in linguistics and logic. In Handbook of logic and language, pages 837–893. Elsevier, 1997.

William R Keller. Nested cooper storage: The proper treatment of quantification in ordinary noun phrases. In Natural language parsing and linguistic theories, pages 432–447. Springer, 1988.

P Kiparsky and C Kiparsky. Fact'in bierwisch and heidolph, eds. Progress in Linguistics, Mouton, The Hague, 1970.

Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1, pages 423–430. Association for Computational Linguistics, 2003.

Angelika Kratzer. Individual-level predicates. The generic book, 125, 1995.

Angelika Kratzer. Severing the external argument from its verb. In Phrase structure and the lexicon, pages 109–137. Springer, 1996.

Angelika Kratzer. The event argument and the semantics of verbs, chapter 2. Manuscript. Amherst: University of Massachusetts, Amherst, MA, 2000.

George Lakoff. Linguistics and natural logic. In Semantics of natural language, pages 545–665. Springer, 1972.

Matthias Lalisse and Ash Asudeh. Distinguishing intersective and non-intersective adjectives in compositional distributional semantics. 2015.

David Lewis. Scorekeeping in a language game. In Semantics from different points of view, pages 172–187. Springer, 1979.

Mike Lewis and Mark Steedman. Combined distributional and logical semantics. Transactions of the Association for Computational Linguistics, 1:179–192, 2013.

Mike Lewis and Mark Steedman. A* CCG parsing with a supertag-factored model. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 990–1000, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1107. URL https://aclanthology.org/D14-1107.

Lindström. First order predicate logic with generalized quantifiers. Theoria, 32(3): 186–195, 1966.

Donald W Loveland, David W Reed, and Debra S Wilson. Satchmore: Satchmo with relevancy. Journal of Automated Reasoning, 14(2):325–351, 1995.

Peter Ludlow. Lf and natural logic. Logical form and language, pages 132–168, 2002.

Bill MacCartney. Natural language inference. PhD dissertation, Stanford University, 2009.

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

Bill MacCartney and Christopher D Manning. 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, 2008.

Bill MacCartney and Christopher D Manning. An extended model of natural logic. In Proceedings of the eighth international conference on computational semantics, pages 140–156. Association for Computational Linguistics, 2009.

Claudia Maienborn. Event semantics. Semantics: An international handbook of natural language meaning, 1:802–829, 2011.

Claudia Maienborn and Martin Schfer. Adverbs and adverbials, volume 2, pages 1390–1420. De Gruyter Mouton, 2011.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In Association for Computational Linguistics (ACL) System Demonstrations, pages 55–60, 2014. URL http://www.aclweb.org/anthology/P/P14/P14-5010.

Rainer Manthey and François Bry. A hyperresolution-based proof procedure and its implementation in prolog. In GWAI-87 11th German Workshop on Artifical Intelligence, pages 221–230. Springer, 1987.

Rainer Manthey and François Bry. Satchmo: a theorem prover implemented in prolog. In International Conference on Automated Deduction, pages 415–434. Springer, 1988.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, Roberto Zamparelli, et al. A sick cure for the evaluation of compositional distributional semantic models. In LREC, pages 216–223, 2014.

Pascual Martínez-Gómez, Koji Mineshima, Yusuke Miyao, and Daisuke Bekki. On-demand injection of lexical knowledge for recognising textual entailment. In

Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 710–720, 2017.

Robert Carlen May. The grammar of quantification. PhD thesis, Massachusetts Institute of Technology, 1978.

John McCarthy. Circumscriptiona form of non-monotonic reasoning. Artificial intelligence, 13(1-2):27–39, 1980.

Louise McNally. Existential sentences, pages 1829 – 1848. De Gruyter Mouton, 2011.

Louise Elizabeth McNally. An interpretation for the english existential construction. 1993.

Merriam-Webster's Online Dictionary. Expletive, 2019. URL https://www.merriam-webster.com/dictionary/expletive. [Online; accessed 30-September-2019].

Line Mikkelsen. 68. Copular clauses, pages 1805 – 1829. De Gruyter Mouton, 2011.

Gary Milsark. Existential sentences in english (doctoral dissertation). Massachusetts Institute of Technology, Cambridge, MA, 1974.

Gary Milsark. Toward an explanation of certain peculiarities of the existential construction in english. 1977.

Richard Montague. The proper treatment of quantification in ordinary english. In Approaches to natural language, pages 221–242. Springer, 1973.

Richard Montague. Formal philosophy: Selected papers, rh thomason, ed, 1974.

Robert C Moore. Semantical considerations on nonmonotonic logic. Artificial intelligence, 25(1):75–94, 1985.

Andrea Moro. Existential sentences and expletive there. The Wiley Blackwell Companion to Syntax, Second Edition, pages 1–26, 2017.

Marcin Morzycki. The lexical semantics of adjectives: More than just scales. Ms., Michigan State University. Draft of a chapter in Modification, a book in preparation for the Cambridge University Press series Key Topics in Semantics and Pragmatics, 2013.

Lawrence S Moss and Michael Wollowski. Natural logic in ai and cognitive science. In MAICS, pages 41–46, 2017.

Andrzej Mostowski. On generalization of quantifiers. Fundamenta Mathematicae, 44: 12–36, 1957.

Alexander PD Mourelatos. Events, processes, and states. Linguistics and philosophy, 2(3):415–434, 1978.

Stefan Müller. HPSG - a synopsis. 01 2014.

Reinhard Muskens. An analytic tableau system for natural logic. In Logic, language and meaning, pages 104–113. Springer, 2010.

Rowan Nairn, Cleo Condoravdi, and Lauri Karttunen. Computing relative polarity for textual inference. Inference in Computational Semantics (ICoS-5), pages 20–21, 2006.

Joakim Nivre. Dependency grammar and dependency parsing. MSI report, 5133 (1959):1–32, 2005.

Toshiyuki Ogihara. Tense, scope and attitude ascription. Dordrecht, NL: Kluwer, 1996.

Toshiyuki Ogihara. Tense and aspect in truth-conditional semantics. Lingua, 117(2): 392–418, 2007.

Toshiyuki Ogihara. Tense, volume 2, pages 1463 – 1484. De Gruyter Mouton, 2011.

Terence Parsons. Events in the Semantics of English, volume 5. MIT press Cambridge, MA, 1990.

Terence Parsons. Underlying states and time travel. Speaking of events, 81:93, 2000.

Barbara Hall Partee. Some structural analogies between tenses and pronouns in english. The Journal of Philosophy, 70(18):601–609, 1973.

Ellie Pavlick. Compositional Lexical Entailment for Natural Language Inference. PhD dissertation, University of Pennsylvania, 2017.

Francis J Pelletier and Allen P Hazen. A history of natural deduction. Logic: A History of Its Central Concepts, 11:341–414, 2012.

Barbara Plank and Gertjan Van Noord. Grammar-driven versus data-driven: which parsing system is more affected by domain shifts? In Proceedings of the 2010 Workshop on NLP and Linguistics: Finding the common ground, pages 25–33, 2010.

Adam Poliak. A survey on recognizing textual entailment as an nlp evaluation. arXiv preprint arXiv:2010.03061, 2020.

Carl Pollard and Ivan A Sag. Information-based syntax and semantics. CSLI lecture notes, 13, 1987.

Carl Pollard and Ivan A Sag. Head-driven phrase structure grammar. University of Chicago Press, 1994.

Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter F Patel-Schneider. Reasoning Web. Semantic Technologies for the Web of Data: 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures, volume 6848. Springer Science & Business Media, 2011.

Dag Prawitz. Natural deduction: a proof-theoretical study. Almquist and Wiksell, 1965.

Arthur N Prior. Past, present and future, volume 154. Clarendon Press, Oxford, 1967.

Allan Ramsay. Genetic nps and habitual vps. In COLING 1992 Volume 1: The 15th International Conference on Computational Linguistics, 1992.

Allan Ramsay. Theorem proving for untyped constructive λ-calculus: implementation and application. Logic Journal of IGPL, 9(1):83–100, 2001.

Allan Ramsay. The complexity of everyday language. Recent Advances in Natural Language Processing V: Selected Papers from RANLP 2007, 309:99, 2009.

Hans Reichenbach. Elements of symbolic logic. 1947.

Raymond Reiter. A logic for default reasoning. Artificial intelligence, 13(1-2):81–132, 1980.

Craige Roberts. Uniqueness in definite noun phrases. Linguistics and philosophy, 26 (3):287–350, 2003.

Bertrand Russell. On denoting. Mind, 14(56):479–493, 1905.

Eddy G Ruys and Yoad Winter. Quantifier scope in formal linguistics. In Handbook of philosophical logic, pages 159–225. Springer, 2011.

Victor Sánchez Valencia. Studies on natural logic and categorial grammar, University of Amsterdam Ph. D. PhD thesis, thesis, 1991.

Marek Sergot. Lecture notes in default logic (reiter), February 2007.

Tim Stowell. What was there before there was there. In Papers from the... Regional Meeting. Chicago Ling. Soc. Chicago, Ill, volume 14, pages 458–471, 1978.

Eric Swanson. Propositional attitudes. Forthcoming in Semantics: An International Handbook of Natural Language Meaning, edited by Claudia Maienborn, Klaus von Heusinger, and Paul Portner, 2010.

Anna Szabolcsi. Scope and binding, volume 2, pages 1605 – 1641. De Gruyter Mouton, 2011.

Joel Tetreault. Empirical Evaluations of Pronoun Resolution. PhD thesis, Rochester, NY, USA, 2005. AAI3156835.

JFAK Van Benthem. Essays in logical semantics. Springer, 1986.

Johan Van Benthem. Questions about quantifiers 1. The Journal of Symbolic Logic, 49(2):443–466, 1984.

Jan Van Eijck. Natural logic for natural language. In International Tbilisi Symposium on Logic, Language, and Computation, pages 216–230. Springer, 2005.

Michiel Van Lambalgen and Fritz Hamm. The proper treatment of events, volume 6. John Wiley & Sons, 2008.

Zeno Vendler. Linguistics in Philosophy. Cornell University Press, 1967.

Klaus Von Heusinger. Reference and representation of pronouns. Pronouns-Grammar and Representation, pages 109–135, 2002.

Jan von Plato. From axiomatic logic to natural deduction. Studia Logica, 102(6): 1167–1184, 2014.

Arnim Von Stechow. Tenses in compositional semantics. The expression of time, 129: 166, 2009.

Dag Westerståhl. Quantifiers in formal and natural languages. In Handbook of philosophical logic, pages 1–131. Springer, 1989.

Dag Westerståhl. Generalized quantifiers: linguistics meets model theory, 2013.

Dag Westersthl. Generalized Quantifiers. In Edward N. Zalta, editor, The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, winter 2019 edition, 2019.

Aaron Steven White, Pushpendre Rastogi, Kevin Duh, and Benjamin Van Durme. Inference is everything: Recasting semantic resources into a unified evaluation framework. In Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 996–1005, 2017.

Wikipedia contributors. Depth-first search, 2020. URL https://en.wikipedia.org/wiki/Depth-first_search#:~:text=Depth%2Dfirst%20search%20(DFS),along%20each%20branch%20before%20backtracking. [Online; accessed 30-September-2020].

Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. arXiv preprint arXiv:1704.05426, 2017.

Edwin Williams. There-insertion. Linguistic inquiry, 15(1):131–153, 1984.

Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, Kentaro Inui, Satoshi Sekine, Lasha Abzianidze, and Johan Bos. Can neural networks understand monotonicity reasoning? arXiv preprint arXiv:1906.06448, 2019.

Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. Transactions of the Association for Computational Linguistics, 2:67–78, 2014.

Annie Zaenen, Lauri Karttunen, and Richard Crouch. Local textual inference: can it be defined or circumscribed? In Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment, pages 31–36, 2005.

Anna Zamansky, Nissim Francez, and Yoad Winter. A natural logicinference system using the lambek calculus. Journal of Logic, Language and Information, 15(3): 273–295, 2006.

Thomas Ede Zimmermann. Scopeless quantifiers and operators. Journal of Philosophical Logic, 22(5):545–561, 1993.

Appendix A

Monotonicity Operators

lexical items and their monotonicity	**inference examples**
every, each, all $(\downarrow)(\uparrow)$	'*every man likes a fast car.*' $\rightarrow$ '*every postman likes a car.*'
bare plurals $(?)(?)$	'the interpretation of bare plurals varies depending on the context – they are similar to universals in 'timeless' contexts and similar to existentials in other contexts, and they display the corresponding monotonicity marking
some, several $(\uparrow)$ $(\uparrow)$	**some(1):** '*some postman loves a pretty woman.*' $\rightarrow$ '*some man likes a woman.*" **some(>1):** '*several/some postmen love Mary.*' $\rightarrow$ '*several/-some men like Mary.*'
a few $(\uparrow)(\uparrow)$	**some(few):** '*a few old men love Mary.*' $\rightarrow$ '*a few men like Mary.*'
no $(\downarrow)(\downarrow)$	'*no man is a fool.*' $\rightarrow$ '*No wise man is a complete fool.*'
not $(\downarrow)$	'*John does not like Mary.*'$\rightarrow$ '*John does not love Mary.*'
most $(\not\uparrow\not\downarrow)(\uparrow)$	'*most men love Mary*' $\rightarrow$ '*most Ment like Mary*'
many$(\uparrow)(\uparrow)$	'*many*' can be interpreted as either a default like '*most*' or an existential like '*a large number of*': we have chosen the latter reading
few $(\not\uparrow\not\downarrow)(\downarrow)$	'*few men like Mary*'$\rightarrow$ '*few men love Mary*'
at least n$(\uparrow)(\uparrow)$	' at least three men slept in the park'$\rightarrow$ '*at least three humans slept*'
at most n$(\downarrow)(\downarrow)$	'*at most four men slept*' $\rightarrow$ '*at most four fat men slept in the park.*'

exactly n(↑)(↑)	'*exactly four old men slept in the park.*' $\not\rightarrow$ '*exactly four men slept.*', '*exactly four men slept in the park.*' $\not\rightarrow$ "*exactly four old men slept.*'

Table A.1: The monotonicity signatures for polarity affecting lexical items

Appendix B

Test Cases

B.1 FraCas Examples

B.1.1 Generalized Quantifiers

Conservativity

fracas-001

P1: An Italian became the worldFLs greatest tenor.

Q: Was there an Italian who became the world's greatest tenor? [Yes]

fracas-002

P1: Every Italian man wants to be a great tenor.

P2: Some Italian men are great tenors.

Q: Are there Italian men who want to be a great tenor? [Yes]

fracas-003

P1: All Italian men want to be a great tenor.

P2: Some Italian men are great tenors.

Q: Are there Italian men who want to be a great tenor? [Yes]

fracas-004

P1: Each Italian tenor wants to be great.

P2: Some Italian tenors are great.

Q: Are there Italian tenors who want to be great? [Yes]

fracas-005

P1: The really ambitious tenors are Italian.

Q: Are there really ambitious tenors who are Italian?	[Yes]

fracas-006

P1: No really great tenors are modest.

Q: Are there really great tenors who are modest?	[No]

fracas-006.1

P1: No tenors are modest

Q: Are there really great tenors who are very modest?	[No]

fracas-007

P1: Some great tenors are Swedish.

Q: Are there great tenors who are Swedish?	[Yes]

fracas-008

P1: Many great tenors are German.

Q: Are there great tenors who are German?	[Yes]

fracas-009

P1: Several great tenors are British.

Q: Are there great tenors who are British?	[Yes]

fracas-010

P1: Most great tenors are Italian.

Q: Are there great tenors who are Italian?	[Yes]

fracas-010.1

P1: Most tenors are Italian.

P2: There are some tenors.

Q: Are there tenors who are Italian?	[Yes]

fracas-010.2

P1: Most great tenors are Italian.

P2: There are some great tenors.

Q: Are there great tenors who are Italian?	[Yes]

fracas-011

P1: A few great tenors sing popular music.

P2: Some great tenors like popular music.

Q: Are there great tenors who sing popular music?	[Yes]

fracas-011.1

P1: A few great tenors sing popular music.

Q: Are there great tenors who sing popular music?	[Yes]

fracas-012

P1: Few great tenors are poor.

Q: Are there great tenors who are poor?	[Unknown]

fracas-013

P1: Both leading tenors are excellent.

P2: Leading tenors who are excellent are indispensable.

Q: Are both leading tenors indispensable?	[Yes]

fracas-014

P1: Neither leading tenor comes cheap.

P2: One of the leading tenors is Pavarotti.

Q: Is Pavarotti a leading tenor who comes cheap?	[No]

fracas-015

P1: At least three tenors will take part in the concert.

Q: Are there tenors who will take part in the concert?	[Yes]
Q: Will there be tenors who will take part in the concert?	[Yes]
Q: Will two tenors take part in the concert?	[Yes]
Q: Will at least two tenors take part in the concert?	[Yes]
Q: Are there three tenors who will take part in the concert?	[Yes]

Q: Will there be three tenors who will take part in the concert?	[Yes]

fracas-016

P1: At most two tenors will contribute their fees to charity.

Q: Are there tenors who will contribute their fees to charity?	[Unknown]
Q: Will three tenors contribute their fees to charity?	[No]
Q: Will at most three tenors contribute their fees to charity?	[Yes]

Monotonicity (upwards on second argument)

fracas-017

P1: An Irishman won the Nobel prize for literature.

Q: Did an Irishman win a Nobel prize?	[Yes]

fracas-018

P1: Every European has the right to live in Europe.

P2: Every European is a person.

P3: Every person who has the right to live in Europe can travel freely within Europe.

Q: Can all Europeans travel freely within Europe?	[Yes]

fracas-018.1

P1: Every European has the right to live.

P2: Every European is a person.

P3: Every person who has the right to live can travel.

Q: Can every Europeans travel?	[Yes]

fracas-018.2

P1: Every European has the right.

P2: Every European is a person.

P3: Every person who has the right can travel.

Q: Can every European travel?	[Yes]

fracas-018.25

P1: Every person has the right.

P2: Every person who has the right can travel.

Q: Can every person travel? [Yes]

fracas-018.3

P1: Every European has the right.

P2: Every European is a person.

P3: Every person who has the right travels.

Q: Does every European travel? [Yes]

fracas-018.4

P1: Every person has the right.

P3: Every person who has the right travels.

Q: Does every person travel? [Yes]

fracas-018.5

P1: Every person sleeps.

P3: Every person who sleeps travels.

Q: Does every person travel? [Yes]

fracas-019

P1: All Europeans have the right to live in Europe.

P2: Every European is a person.

P3: Every person who has the right to live in Europe can travel freely within Europe.

Q: Can all Europeans travel freely within Europe? [Yes]

fracas-020

P1: Each European has the right to live in Europe.

P2: Every European is a person.

P3: Every person who has the right to live in Europe can travel freely within Europe.

Q: Can each European travel freely within Europe? [Yes]

fracas-021

P1: The residents of member states have the right to live in Europe.

P2: All residents of member states are individuals.

P3: Every individual who has the right to live in Europe can travel freely within Europe.

Q: Can the residents of member states travel freely within Europe? [Yes]

fracas-021.01

P1: The residents of the member states have the right to live in Europe.

P2: All residents of the member states are individuals.

P3: Every individual who has the right to live in Europe can travel freely within Europe.

Q: Can the residents of the member states travel freely within Europe? [Yes]

fracas-021.1

P1: The residents of the states have the right.

P2: All residents of the states are individuals.

P3: Every individual who has the right can travel.

Q: Can the residents of the states travel? [Yes]

fracas-021.11

P1: The residents of states have the right.

P2: All residents of the states are individuals.

P3: Every individual who has the right can travel.

Q: Can the residents of states travel? [Yes]

fracas-021.111

P1: The residents have the right.

P2: All residents are individuals.

P3: Every individual who has the right can travel.

Q: Can the residents travel? [Yes]

fracas-021.12

P1: The residents of states have the right.

P2: The residents of states are individuals.

P3: Every individual who has the right can travel

Q: Can the residents of states travel? [Yes]

fracas-021.15

P1: The residents of states have the right to live in Europe.

P2: All residents of states are individuals.

P3: Every individual who has the right to live in Europe can travel freely within Europe.

Q: Can the residents of states travel freely within Europe? [Yes]

fracas-021.16

P1: The residents of a state have the right to live in Europe.

P2: All residents of a state are individuals.

P3: Every individual who has the right to live in Europe can travel freely within Europe.

Q: Can the residents of a state travel freely within Europe? [Yes]

fracas-021.2

P1: The residents have the right to live in Europe.'

P2: All residents are individuals.

P3: Every individual who has the right to live in Europe can travel freely within Europe.

Q: Can the residents travel freely within Europe? [Yes]

fracas-022

P1: No delegate finished the report on time.

Q: Did no delegate finish the report? [Unknown]

fracas-022.1

P1: No delegate finished the report.

Q: Did no delegate finish the report on time? [Yes]

Q: Did some delegate finish the report on time? [No]

fracas-023

P1: Some delegates finished the survey on time.

Q: Did some delegates finish the survey?	[Yes]

fracas-024

P1: Many delegates obtained interesting results from the survey.

Q: Did many delegates obtain results from the survey?	[Yes]

fracas-025

P1: Several delegates got the results published in major national newspapers.

Q: Did several delegates get the results published?	[Yes]

fracas-025.1

P1: Several delegates got the results published.

Q: Did several delegates get the results published?	[Yes]

fracas-026

P1: Most Europeans are resident in Europe.

P: All Europeans are people.

P3: All people who are resident in Europe can travel freely within Europe.

Q: Can most Europeans travel freely within Europe?	[Yes]

fracas-026.1

P1: Most Europeans are resident in Europe.

P: All Europeans are people.

P3: All people who are resident in Europe can travel freely within Europe.

P: John is a European.

Q: Can John travel freely within Europe?	[Yes]

fracas-027

P1: A few committee members are from Sweden.

P2: All committee members are people.

P3: All people who are from Sweden are from Scandinavia.

Q: Are at least a few committee members from Scandinavia?	[Yes]
Q: Are a few committee members from Scandinavia?	[Yes]

fracas-027.1

P1: A few committee members are from Sweden.

Q: At least a few committee members from Sweden?	[Yes]
Q: Are a few committee members from Sweden?	[Yes]
Q: Are there a few committee members from Sweden?	[Yes]

fracas-027.2

P1: All people who are from Sweden are from Scandinavia.

P2: There are some people from Sweden.

Q: Are there some people from Scandinavia?	[Yes]

fracas-027.2

P1: All people who are from Sweden are from Scandinavia.

P2: There are some people from Sweden.

Q: Are there some people from Scandinavia?	[Yes]

fracas-028

P1: Few committee members are from Portugal.

P2: All committee members are people.

P3: All people who are from Portugal are from southern Europe.

Q: Are there few committee members from southern Europe?	[Unknown]

fracas-029

P1: Both commissioners used to be leading businessmen.

Q: Did both commissioners used to be businessmen?	[Yes]

fracas-030

P1: Neither commissioner spends a lot of time at home.

Q: Does neither commissioner spend time at home?	[Unknown]

fracas-031

P1: At least three commissioners spend a lot of time at home.

Q: Do at least three commissioners spend time at home? [Yes]

fracas-032

P1: At most ten commissioners spend a lot of time at home.

Q: Do at most ten commissioners spend time at home? [Unknown]

Monotonicity (downwards on second argument)

fracas-033

P1: An Irishman won a Nobel prize.

Q: Did an Irishman win the Nobel prize for literature? [Unknown]

fracas-034

P1: Every European can travel freely within Europe.

P2: Every European is a person.

P3: Every person who has the right to live in Europe can travel freely within Europe.

Q: Does every European have the right to live in Europe? [Unknown]

fracas-035

P1: All Europeans can travel freely within Europe.

P2: Every European is a person.

P3: Every person who has the right to live in Europe can travel freely within Europe.

Q: Does every European have the right to live in Europe? [Unknown]

fracas-036

P1: Each European can travel freely within Europe.

P2: Every European is a person.

P3: Every person who has the right to live in Europe can travel freely within Europe.

Q: Does each European have the right to live in Europe? [Unknown]

fracas-037

fracas-038

P1: No delegate finished the report.

Q: Did any delegate finish the report on time?	[no]

fracas-039

P1: Some delegates finished the survey.

Q: Did some delegates finish the survey on time?	[Unknown]

fracas-040

P1: Many delegates obtained results from the survey.

Q: Did many delegates obtain interesting results from the survey?	[Unknown]

fracas-041

P1: Several delegates got the results published.

Q: Did several delegates get the results published in major national newspapers?	[Unknown]

fracas-042

P1: Most Europeans can travel freely within Europe.

P2: All Europeans are people.

P3: All people who are resident in Europe can travel freely within Europe.

Q: Are most Europeans resident in Europe?	[Unknown]

fracas-043

P1: A few committee members are from Scandinavia.

P2: All committee members are people.

P3: All people who are from Sweden are from Scandinavia.

Q: Are at least a few committee members from Sweden?	[Unknown]

fracas-044

P1: Few committee members are from southern Europe.

P2: All committee members are people.

P3: All people who are from Portugal are from southern Europe.

Q: Are there few committee members from Portugal?	[yes]

fracas-045

P1: Both commissioners used to be businessmen.	
Q: Did both commissioners used to be leading businessmen?	[Unknown]

fracas-046

P1: Neither commissioner spends time at home.	
Q: Does either commissioner spend a lot of time at home?	[no]

fracas-047

P1: At least three commissioners spend time at home.	
Q: Do at least three commissioners spend a lot of time at home?	[Unknown]

fracas-048

P1: At most ten commissioners spend time at home.	
Q: Do at most ten commissioners spend a lot of time at home?	[yes]

Monotonicity (upwards on first argument)

fracas-049

P1: A Swede won a Nobel prize. P2: Every Swede is a Scandinavian.	
Q: Did a Scandinavian win a Nobel prize?	[yes]

fracas-050

P1: Every Canadian resident can travel freely within Europe. P2: Every Canadian resident is a resident of the North American continent.	
Q: Can every resident of the North American continent travel freely within Europe?	[Unknown]

fracas-051

P1: All Canadian residents can travel freely within Europe. P2: Every Canadian resident is a resident of the North American continent.	

Q: Can all residents of the North American continent travel freely within Europe?	[Unknown]
fracas-052 P1: Each Canadian resident can travel freely within Europe. P2: Every Canadian resident is a resident of the North American continent.	
Q: Can each resident of the North American continent travel freely within Europe?	[Unknown]
fracas-053 P1: The residents of major western countries can travel freely within Europe. P2: All residents of major western countries are residents of western countries.	
Q: Do the residents of western countries have the right to live in Europe?	[Unknown]
fracas-054 P1: No Scandinavian delegate finished the report on time.	
Q: Did any delegate finish the report on time?	[Unknown]
fracas-055 P1: Some Irish delegates finished the survey on time.	
Q: Did any delegates finish the survey on time?	[yes]
fracas-056 P1: Many British delegates obtained interesting results from the survey.	
Q: Did many delegates obtain interesting results from the survey?	[Unknown]
fracas-057 P1: Several Portuguese delegates got the results published in major national newspapers.	

Q: Did several delegates get the results published in major national newspapers?	[yes]

fracas-058

P1: Most Europeans who are resident in Europe can travel freely within Europe.	
Q: Can most Europeans travel freely within Europe?	[Unknown]

fracas-059

P1: A few female committee members are from Scandinavia.	
Q: Are at least a few committee members from Scandinavia?	[yes]

fracas-060

P1: Few female committee members are from southern Europe.	
Q: Are few committee members from southern Europe?	[Unknown]

fracas-061

P1: Both female commissioners used to be in business.	
Q: Did both commissioners used to be in business?	[undef]

fracas-062

P1: Neither female commissioner spends a lot of time at home.	
Q: Does either commissioner spend a lot of time at home?	[undef]

fracas-063

P1: At least three female commissioners spend time at home.	
Q: Do at least three commissioners spend time at home?	[yes]

fracas-064

P1: At most ten female commissioners spend time at home.	
Q: Do at most ten commissioners spend time at home?	[yes]

Monotonicity (downwards on first argument)

fracas-065

P1: A Scandinavian won a Nobel prize.

P2: Every Swede is a Scandinavian.

Q: Did a Swede win a Nobel prize?	[Unknown]

fracas-066

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.

Q: Can every Canadian resident travel freely within Europe?	[yes]

fracas-067

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.

Q: Can all Canadian residents travel freely within Europe?	[yes]

fracas-068

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.

Q: Can each Canadian resident travel freely within Europe?	[yes]

fracas-069

P1: The residents of western countries can travel freely within Europe.

P2: All residents of major western countries are residents of western countries.

Q: Do the residents of major western countries have the right to live in Europe?	[yes]

fracas-070

P1: No delegate finished the report on time.

Q: Did any Scandinavian delegate finish the report on time?	[no]

fracas-071

P1: Some delegates finished the survey on time.

Q: Did any Irish delegates finish the survey on time?	[Unknown]

fracas-072

P1: Many delegates obtained interesting results from the survey.

Q: Did many British delegates obtain interesting results from the survey?	[Unknown]

fracas-073

P1: Several delegates got the results published in major national newspapers.

Q: Did several Portuguese delegates get the results published in major national newspapers?	[Unknown]

fracas-074

P1: Most Europeans can travel freely within Europe.

Q:Can most Europeans who are resident outside Europe travel freely within Europe?	[Unknown]

fracas-075

P1: A few committee members are from Scandinavia.

Q:Are at least a few female committee members from Scandinavia?	[Unknown]

fracas-076

P1: Few committee members are from southern Europe.

Q:Are few female committee members from southern Europe?	[yes]

fracas-077

P1: Both commissioners used to be in business.

Q:Did both female commissioners used to be in business?	[undef]

fracas-078

P1: Neither commissioner spends a lot of time at home.	
Q:Does either female commissioner spend a lot of time at home?	[undef]

fracas-079

P1: At least three commissioners spend time at home.	
Q:Do at least three male commissioners spend time at home?	[unknown]

fracas-080

P1: At most ten commissioners spend time at home.	
Q:Do at most ten female commissioners spend time at home?	[yes]

B.1.2 Attitudes

Epistemic, Intentional and Reportive Attitudes

fracas-334

P1: Smith knew that ITEL had won the contract in 1992.	
Q: Did ITEL win the contract in 1992?	[Yes]

fracas-335

P1: Smith believed that ITEL had won the contract in 1992.	
Q: Did ITEL win the contract in 1992?	[unknown]

fracas-336

P1: ITEL managed to win the contract in 1992.	
Q: Did ITEL win the contract in 1992?	[Yes]

fracas-337

P1: ITEL tried to win the contract in 1992.	
Q: Did ITEL win the contract in 1992?	[unknown]

fracas-338

P1: It is true that ITEL won the contract in 1992.	
Q: Did ITEL win the contract in 1992?	[Yes]

fracas-339

P1: It is false that ITEL won the contract in 1992.

Q: Did ITEL win the contract in 1992? [No]

Preceptive Attitudes: "See" with Bare Infinitive Complements

fracas-340

P1: Smith saw Jones sign the contract.

P2: If Jones signed the contract, his heart was beating.

Q: Did Smith see Jones' heart beat? [unknown]

fracas-341

P1: Smith saw Jones sign the contract.

P2: When Jones signed the contract, his heart was beating.

Q: Did Smith see Jones' heart beat? [unknown]

fracas-342

P1: Smith saw Jones sign the contract.

Q: Did Jones sign the contract? [Yes]

fracas-343

P1: Smith saw Jones sign the contract.

P2: Jones is the chairman of ITEL.

Q: Did Smith see the chairman of ITEL sign the contract? [Yes]

fracas-344

P1: Helen saw the chairman of the department answer the phone.

P2: The chairman of the department is a person.

Q: Is there anyone whom Helen saw answer the phone? [Yes]

fracas-345

P1: Smith saw Jones sign the contract and his secretary make a copy.

Q: Did Smith see Jones sign the contract? [Yes]

fracas-347

P1: Smith saw Jones sign the contract or cross out the crucial clause.	
Q: Did Smith either see Jones sign the contract or see Jones cross out the crucial clause?	[Yes]

B.2 Karttunen's Examples of Implicatives and Factives

fracas-k00

P1: John believes a fat old man loves Mary.	
Q: does John believe a human likes Mary?	[Yes]

fracas-k01

P1: John does not believe a human likes Mary	
Q: does John not believe a man loves Mary?	[Yes]

fracas-k02

P1: John believes every man loves Mary.	
Q: does John believe every fat old man likes Mary?	[Yes]

fracas-k03

P1: John believes every man loves Mary.	
Q: does John believe every fat old man with a big nose likes Mary?	[Yes]

fracas-k04

P1: John managed to eat a ripe peach.	
Q: did John manage to eat a peach?	[Yes]

fracas-k05

P1: John did not manage to eat a peach.	
Q: did John manage to eat a ripe peach?	[No]

fracas-k06

P1: John managed to eat a peach.	
Q: did John eat a peach?	[Yes]

fracas-k07

P1: John did not manage to eat a peach.	
Q: did John eat a peach?	[No]

fracas-k08

P1: John failed to eat a peach.	
Q: did John eat a peach?	[No]

fracas-k09

P1: John did not fail to eat a peach.	
Q: did John eat a peach?	[Yes]

fracas-k10

P1: John used to beat his wife.	
Q: did John beat his wife?	[Yes]
Q: does John beat his wife?	[No]

fracas-k11

P1: John has stopped beating his wife.	
Q: did John beat his wife?	[Yes]
Q: does John beat his wife?	[No]

fracas-k12

P1: John has not stopped beating his wife.	
Q: did John beat his wife?	[Yes]
Q: does John beat his wife?	[Yes]

fracas-k13

P1: I know John ate a peach.	
Q: did John eat a peach?	[Yes]

fracas-k14

P1: I know John managed to eat a peach.	
Q: did John eat every ripe peach?	[Yes]

fracas-k15

P1:I know John managed to eat every peach.

Q: did John eat a peach? [Yes]

fracas-k16

P1:I doubt that Mary likes John.

Q: I doubt that Mary loves John? [Yes]

fracas-k17

P1: I do not doubt that Mary loves John.

Q:I do not doubt that Mary likes John? [Yes]

fracas-k18

P1: I doubt that John doubts that Mary loves John.

Q:I doubt that John doubts that Mary likes John? [Yes]

fracas-k19

P1: I doubt that John doubts that Mary loves him.

Q: I doubt that John doubts that Mary likes him? [Yes]

fracas-k20

P1: I doubt that John doubts that Mary likes him.

Q:I doubt that John doubts that Mary loves him? [Unknown]

fracas-k21

P1: I know John did not sleep.

Q: did John sleep? [No]

fracas-k22

P1:I know John did not manage to sleep.

Q: did John sleep? [No]

fracas-k23

P1: ITEL failed to win the contract in 1992.

Q: did ITEL win the contract in 1992? [No]

fracas-k24a

P1: I believe a man slept.	
Q: do I doubt a man slept?	[No]
Q: do I doubt a fat man slept?	[Unknown]

fracas-k24b

P1: I doubt a man slept.	
Q: do I believe a man slept?	[No]
Q: do I believe a fat man slept?	[No]

fracas-k24c

P1: I believe a fat man slept.	
Q: do I doubt that a fat man slept?	[No]
Q: do I doubt a man slept?	[No]

fracas-k24d

P1: I doubt a fat man slept.	
Q: do I believe that a fat man slept?	[No]
Q: do I believe a man slept?	[Unknown]

Appendix C

Results of Running The Test Cases

C.1 Results of Running FraCas Section 1 and 9 Examples

Fracas ID	Gold answer	CSATCHMO+ answer	Comments
fracas-001	Yes	Yes	
fracas-002	Yes	Yes	
fracas-003	Yes	Yes	
fracas-004	Yes	Yes	
fracas-005	Yes	Yes	
fracas-006	No	No	
fracas-007	Yes	Yes	
fracas-008	Yes	Yes	
fracas-009	Yes	Yes	
fracas-010	Yes	Unknown	You cannot infer that there are any great tenors from statements about '*most great tenors*' any more than you can from statements about '*all great tenors*'
fracas-011	Yes	Yes	
fracas-012	Undef**	Unknown	
fracas-013	Yes	Yes	
fracas-014	No	No	
fracas-015	Yes	Unkown	We do not know if there **are** any tenors **now** who will take part in the concert **later**

fracas-016	Undef**	Unknown	
fracas-017	Yes	Yes	
fracas-018	Yes	Yes	
fracas-019	Yes	Yes	
fracas-020	Yes	Yes	
fracas-021	Yes	Yes	
fracas-022	Unknown	Unknown	
fracas-023	Yes	Yes	
fracas-024	Yes	Yes	
fracas-025	Yes	Yes	
fracas-026	Yes	Yes	
fracas-027	Yes	Yes	
fracas-028	Unknown	Unknown	
fracas-029	Yes	Yes	
fracas-030	Unknown	Unknown	
fracas-031	Yes	Yes	
fracas-032	Unknown	Unknown	
fracas-033	Unknown	Unknown	
fracas-034	Unknown	Unknown	
fracas-035	Unknown	Unknown	
fracas-036	Unknown	Unknown	
fracas-037	Unknown	Unknown	
fracas-038	No	No	
fracas-039	Unknown	Unknown	
fracas-040	Unknown	Unknown	
fracas-041	Unknown	Unknown	
fracas-042	Unknown	Unknown	
fracas-043	Unknown	Unknown	
fracas-044	Yes	Yes	
fracas-045	Unknown	Unknown	
fracas-046	No	No	
fracas-047	Unknown	Unknown	
fracas-048	Yes	Yes	
fracas-049	Yes	Yes	
fracas-050	Unknown	Unknown	
fracas-051	Unknown	Unknown	

fracas-052	Unknown	Unknown	
fracas-053	Unknown	Unknown	
fracas-054	Unknown	Unknown	
fracas-055	Yes	Yes	
fracas-056	Unknown	Unknown	
fracas-057	Yes	Yes	
fracas-058	Unknown	Unknown	
fracas-059	Yes	Yes	
fracas-060	Unknown	Unknown	
fracas-061	Undef**	Unknown	
fracas-062	Undef**	Unknown	
fracas-063	Yes	Yes	
fracas-064	Unknown	Unknown	
fracas-065	Unknown	Unknown	
fracas-066	Yes	Yes	
fracas-067	Yes	Yes	
fracas-068	Yes	Yes	
fracas-069	Yes	Unknown	
fracas-070	No	No	
fracas-071	Unknown	Unknown	
fracas-072	Unknown	Unknown	
fracas-073	Unknown	Unknown	
fracas-074	Unknown	Yes	We get '*Yes with defaults*' as the answer to this one, because we do not mark the polarity of '*most*' right.
fracas-075	Unknown	Unknown	
fracas-076	Yes	Yes	
fracas-077	Undef**	Unknown	
fracas-078	Undef**	Unknown	
fracas-079	Unknown	Unknown	
fracas-080	Yes	Yes	
fracas-334	Yes	Unknown	It is not clear why we get this one wrong. The mismatch between the past tenses in P1 and Q and the present tense in P2 is potentially an issue, but there is something else wrong as well.

fracas-335	Unknown	Unknown	
fracas-336	Yes	Yes	
fracas-337	Unknown	Unknown	
fracas-338	Yes	Yes	
fracas-339	No	No	
fracas-340	Unknown	Unknown	
fracas-341	Unknown	Unknown	
fracas-342	Yes	Yes	
fracas-343	Yes	Yes	
fracas-344	Yes	Yes	
fracas-345	Yes	Unknown	We do not deal with conjunctions and disjunctions
fracas-346	Yes	Unknown	We do not deal with conjunctions and disjunctions

C.2 Results of Running Karttunen's Examples

Fracas ID	**Gold answer**	**CSATCHMO+ answer**	**LangPro answer**
fracas-k00	Yes	Yes	Unknown
fracas-k01	Yes	Yes	Unknown
fracas-k02	Yes	Yes	Yes
fracas-k03	Yes	Yes	Unknown
fracas-k04	Yes	Yes	Yes
fracas-k05	No	No	No
fracas-k06	Yes	Yes	Yes
fracas-k07	No	No	No
fracas-k08	No	No	Unknown
fracas-k09	Yes	Yes	Unknown
fracas-k10	Yes/No	Yes/No	Unknown/Unknown
fracas-k11	Yes/No	Yes/No	Unknown/Unknown
fracas-k12	Yes/Yes	Yes/Yes	Unknown/Unknown
fracas-k13	Yes	Yes	Yes
fracas-k14	Yes	Yes	Yes
fracas-k15	Yes	Yes	Unknown
fracas-k16	Yes	Yes	Unknown

fracas-k17	Yes	Yes	Unknown
fracas-k18	Yes	Yes	Yes
fracas-k19	Yes	Yes	Yes
fracas-k20	Unknown	Unknown	Unknown
fracas-k21	No	No	No
fracas-k22	No	No	No
fracas-k23	No	No	Unknown
fracas-k24a	No/Unknown	No/Unknown	Unknown/Unknown
fracas-k24b	No/No	No/No	Unknown/Unknown
fracas-k24c	No/No	No/No	Unknown/Unknown
fracas-k24d	No/Unknown	No/Unknown	Unknown/Unknown

Index