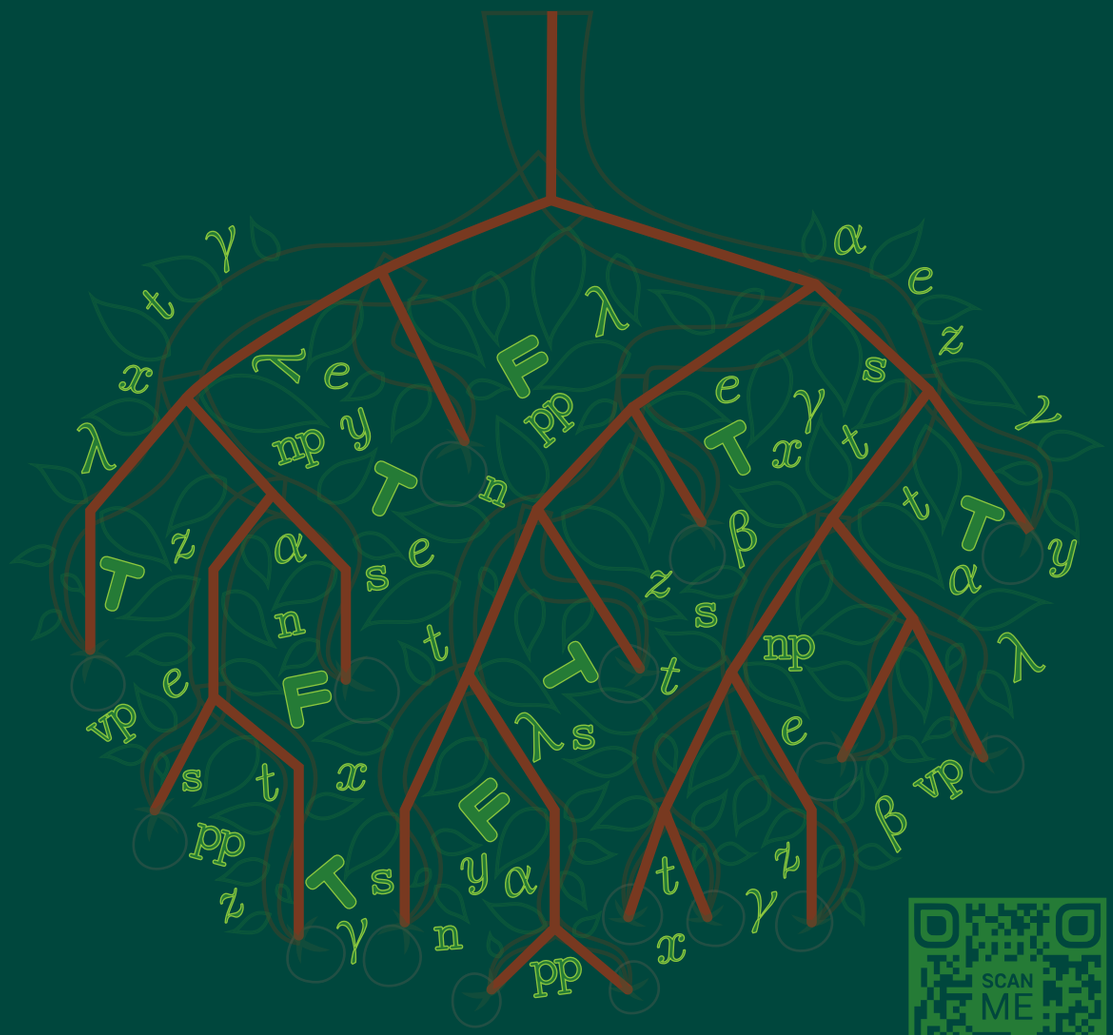
In the title of this thesis, a *proof system* refers to a system that carries out formal proofs (e.g., of theorems). Logicians would interpret a *system* as a symbolic calculus while computer scientists might understand it as a computer program. Both interpretations are fine for the current purpose as the thesis develops a proof calculi and a computer program based on it. What makes the proof system *natural* is that it operates on formulas with a natural appearance, i.e. resembling natural language phrases. This is contrasted to the *artificial* formulas logicians usually use. Put differently, the natural proof system is designed for a *natural logic*, a logic that has formulas similar to natural language sentences. The natural proof system represents a further development of an analytic tableau system for natural logic (Muskens, 2010). The implementation of the system acts as a theorem prover for wide-coverage natural language sentences. For instance, it can prove that *not all PhD theses are interesting* entails *some dissertations are not interesting*. On certain textual entailment datasets, the prover achieves results competitive to the state-of-the-art.



ISBN 978-94-6299-494-2

SCAN ME

A Natural Proof System for Natural Language

Lasha Abzianidze

A NATURAL PROOF SYSTEM FOR NATURAL LANGUAGE

Lasha Abzianidze

A Natural Proof System for Natural Language

ISBN 978-94-6299-494-2

Cover & inner design by Lasha Abzianidze
Used item: leaf1 font by Rika Kawamoto

Printed and bound by Ridderprint BV, Ridderkerk

A Natural Proof System for Natural Language

Proefschrift

ter verkrijging van de graad van doctor aan Tilburg University op gezag van de rector magnificus, prof.dr. E.H.L. Aarts, in het openbaar te verdedigen ten overstaan van een door het college voor promoties aangewezen commissie in de aula van de Universiteit op vrijdag 20 januari 2017 om 14.00 uur door

Lasha Abzianidze

geboren op 24 januari 1987 te Tbilisi, Georgië

Promotiecommissie

Promotor	Prof. dr. J.M. Sprenger
Copromotor	Dr. R.A. Muskens
Overige leden	Prof. dr. Ph. de Groote
	Dr. R. Moot
	Prof. dr. L.S. Moss
	Prof. dr. Y. Winter

Acknowledgments

During working on this dissertation, there were many people who helped and supported me, both inside and outside the academic life. Here I would like to thank them.

First, my sincere thanks go to my supervisors Jan and Reinhard. I learned a lot from them during this period. Their joint feedback and guidance were vital for my research. This dissertation would not have been possible without their input. Many thanks go to Reinhard who believed that I was a suitable candidate for his innovative research program. He continuously provided me with crucial advice and suggestions during the research. I thank him for believing in me and giving me a freedom to choose a more exciting and demanding research direction towards wide-coverage semantics. Jan gave me valuable and needed support in the later stage of the project. His comments and suggestions have significantly improved the presentation in the thesis. I also benefited from his advice about job applications. I am grateful to him.

I feel very lucky to have so much expertise and leading researchers of my field on the committee. I am thankful to Philippe de Groote, Richard Moot, Larry Moss and Yoad Winter for that. Additionally, I am indebted to Philippe for his support during my LCT master study, and I want express my gratitude to Larry whose comments and opinion on my work has been inspiring.

Special thanks go to my office mate Janine and my close friend Irakli who agreed to be my paranymphs. I also thank Janine for preparing delicious cakes and sweets for me. Irakli has been a friend I can rely on any time. I appreciate it.

I acknowledge the valuable benefit and feedback I got from participating in several ESSLLI schools, attending and presenting at several workshops and conferences, among them LENLS, WoLLIC, EMNLP, TbiLLC, Amsterdam Colloquium, ACL and *SEM. The local TiLPS seminars were very helpful, where I presented early results of my research and widened my knowledge of logic, language and philosophy.

My working environment and stay in Tilburg were enjoyable due to my colleagues at TiLPS: Alessandra, Alfred, Barbara, Bart, Chiara, Colin, Dominik, Georgi, Jan, Janine, Jasmina, Jun, Silvia, Machteld, Matteo, Michal, Naftali, Reinhard, and Thomas. Thank you guys; with you I enjoyed playing footy in the corridor, having drinks, dinners or parties, watching football, going on TiLPS excursions, or simply spending time together. I also want to thank my close friend Sandro, who was also doing his PhD meanwhile. The (skype) conversations with him were helpful, encouraging and entertaining for me.

Many thanks go to my family on the other side of the continent. Especially I want to thank my mother, who always did her best to support and create conditions for me to feel comfortable and to concentrate on my studies. I am deeply grateful to her for everything.

Finally and mostly, I would like to thank *mu kallis* Eleri for her continuous and endless support, patience and unconditional love during this period. She bared me when I was obsessed with research and writing. Her help and care have been indispensable for me. I cannot thank her enough.

Abstract

We humans easily understand semantics of natural language text and make inferences based on it. But how can we make machines reason and carry out inferences on the text? The question represents a crucial problem for several research fields and is at the heart of the current thesis.

One of the most intuitive approaches to model reasoning in natural language is to translate semantics of linguistic expressions into some formal logic that comes with an automated inference procedure. Unfortunately, designing such automatized translation turns out to be almost as hard as the initial problem of reasoning.

In this thesis, following Muskens (2010), we develop a model for natural reasoning that takes a proof theoretic stand. Instead of a heavy translation procedure, we opt for a light-weight one. Logical forms obtained after the translation are *natural* in the sense that they come close to the original linguistic expressions. But at the same time, they represent terms of higher-order logic. In order to reason over such logical forms, we extend a tableau proof system for type theory by Muskens. An obtained *natural* tableau system employs inference rules specially tailored for various linguistic constructions.

Since we employ the logical forms close to linguistic expressions, our approach contributes to the project of *natural logic*. Put differently, we employ the higher-order logic disguised as natural logic. Due to this hybrid nature, the proof system can carry out both shallow and deep reasoning over multiple-premised arguments. While doing so, a main vehicle for shallow reasoning is monotonicity calculus.

In order to evaluate the natural proof system, we automatize it as a theorem prover for wide-coverage natural language text. In particular, first the logical forms are obtained from syntactic trees of linguistic expressions, and then they are processed by the prover. Despite its simple architecture, on certain textual entailment benchmarks the prover obtains high competitive results for both shallow and deep reasoning. Notice that this is done while it employs only WordNet as a knowledge base. The theorem prover also represents the first wide-coverage system for natural logic which can reason over multiple propositions at the same time.

The thesis makes three major contributions. First, it significantly extends the natural tableau system for wide-coverage natural reasoning. After extending the type system and the format of tableau entries (Chapter 2), we collect a plethora of inference rules for a wide range of constructions (Chapter 4). Second, the thesis proposes a procedure to obtain the logical forms from syntactic derivations of linguistic expressions (Chapter 3). The procedure is anticipated as a useful tool for wide-coverage compositional semantic analysis. Last, based on the natural tableau system, the theorem prover for natural language is implemented (Chapter 5,6). The prover represents a state-of-the-art system of natural logic.

Contents

Chapter 1

Introduction

Inferring natural language sentences from a text is the central problem for the thesis and is known as Natural Language Inference (NLI). The chapter starts with the introduction to NLI and presents a task, called Recognizing Textual Entailment (RTE), which was designed by the Natural Language Processing (NLP) community in order to tackle the NLI problem (§ 1.1). Before presenting our approach to the RTE task in the next chapters, first we describe an intuitive model for solving the task, and then we briefly overview some existing approaches to textual entailment ranging from shallow to deep approaches (§ 1.2). Later we focus on the research line our approach contributes to. In particular, we introduce the project of natural logic and describe its specialty—monotonicity reasoning (§ 1.3). Next, we discuss a natural logic approach to textual entailment and present the work by MacCartney and Manning (2007), which suggested the first mature application of natural logic to RTE (§ 1.4). In the final section, we outline the rest of the chapters which gradually present our natural logic-based approach to RTE.

1.1 Natural language inference

In a broad sense, Natural Language Inference (NLI) is a process of inferring a (target) natural language text T from the meaning of another (source) natural language text S. In practice, a text can range from a sequence of sentences to a single sentence, or even to a phrase. We say that S *infers* T if and only if it is highly probable that T is true whenever S is true. We also could define the inference relation as human inference: most humans accept T as true, whenever S happens to be true. The key is that in both cases the notion of inference is imprecise and depends on factors such as probability, acceptance, and human understanding of S and T. For a given source text, one can infer several facts expressed in terms of natural language. For example, consider the sentence in (1) as a source text. Then the facts expressed by the sentences in (1a) and (1b) can be inferred from it. While (1a) is necessarily true when (1) is true, (1b) is highly probable as usually a buyer club violates some rules concerning a transfer and therefore pays a fine. On the other hand, the sentence in (1c) expresses the meaning which is not inferred from (1).

Barcelona football club agreed to pay a €5.5m fine over the transfer of Brazil international Neymar in 2013 (1)

The football club Barcelona agreed to pay a penalty for a transfer (1a)

Neymar signed for the football club Barcelona in 2013 (1b)

Neymar is the first Brazilian international in FC Barcelona (1c)

The study of NLI attempts to answer several questions: (i) how do humans process and reason over natural language text? (ii) how shall we automatize the reasoning? (iii) which representation of linguistic semantics is suitable for modeling NLI? An answer to one of these questions can be a key while answering the rest of the questions. Since the current work focuses on the latter two questions, we present the problem of NLI from the perspectives of Natural Language Processing (NLP) and Artificial Intelligence.

The tasks concerning NLI can be seen "as the best way of testing an NLP system's semantic capacity" (Cooper et al., 1996, p. 63). One straightforward task of NLI is to list the sentences that are inferred from a given source text. Unfortunately, from an evaluation perspective, the task is ill-defined due to diversity of inferred facts and natural language variability. For any source text, it is unclear how to define a finite list of correct (i.e. gold) inferred sentences. Moreover, for each candidate sentence its meaning can be expressed in several different ways due to language variability. For example, given the sentence in (1), it is unclear which and how many sentences an NLP system should infer from it. This indeterminacy makes it difficult to evaluate answers of NLP systems. Another alternative NLI task is Question Answering (QA): based on a source text, to answer a question with YES, NO or UNKNOWN (Cooper et al., 1996). For instance, to answer the question in (2) based on the information provided by (1). With the help of yes-no-unknown questions, we avoid the indeterminacy caused by natural language variability—an NLP system does not need to generate natural language expressions.

Did the football club Barcelona agree to pay a penalty for a transfer? (2)

a €5.5m fine over the transfer of Brazil international Neymar (3)

But lengthy yes-no-unknown questions are unusual in natural language usage. Moreover, such questions are asked in context of declarative sentences and are not applicable to non-sentential phrases, e.g., the noun phrase in (3). In order to free an NLI task from interrogative sentences, the NLP community came up to an NLI task that contrasts semantics of two declarative texts.

The task of Recognizing Textual Entailment (RTE) was introduced by Dagan et al. (2006) and it attempts to evaluate an NLP systems competence in NLI. The objective in the task is to guess an entailment (i.e. inference) relation[1] from a text T to a hypothesis H. The text T is a sentence or a set of sentences while the hypothesis H is a single sentence or a phrase. In order to create a benchmark for the RTE task, first text-hypothesis pairs are collected manually or semi-automatically, and then each pair is annotated with entailment relations by humans. Usually canonical entailment relations are ENTAILMENT (i.e. YES), CONTRADICTION (i.e. NO) and NEUTRAL (i.e. UNKNOWN) depending on whether T

[1] In logic, inference is more general notion than entailment: a proposition Q can be inferred from a proposition P but not entailed. For example, Q can be inferred from P if Q represents the best explanation for P (the latter called as *abductive inference*). However, at the same time, it can be the case that Q is not entailed from P: there is a situation (though with little probability) where P is true and Q is false. So, entailment is one of the sorts of inference. When concerning NLI, both notions are often used as synonyms since entailment losses its strict logical sense and becomes *soft*.

entails (i.e. infers), contradicts or is neutral to H.[2]According to the RTE guidelines, verb tense and aspect issues must be ignored during the annotation (Dagan et al., 2006). In the end, only those text-hypothesis pairs with high annotation agreement are included in the RTE benchmark dataset. If an RTE system's guesses resemble the human annotations (also referred as the *gold labels*), then it is assumed that the system can emulate human-like reasoning on the benchmark dataset.[3] An example of text-hypothesis pair, i.e. an RTE problem, from the first RTE dataset (Dagan et al., 2006) is given below, where the problem has its data-specific ID and an ENTAILMENT gold label.

Green cards are becoming more difficult to obtain
Green card is now difficult to receive
GOLD: ent; RTE1-62

When judging textual entailment pairs, humans unintentionally employ their knowledge of the world and natural language. This knowledge is later reflected by the gold labels. In order to achieve high performance on the RTE task, a system is expected to get the knowledge presupposed by humans. For instance, we have already mentioned the world knowledge that contributes to the entailment of (1b) from (1). Awareness of synonymous meanings of "*obtain*" and "*receive*" in the context of "*green card*" (RTE1-62) represents knowledge of natural language. Since the above mentioned knowledge is vast and almost each textual entailment problem requires some of it, RTE systems are always hungry for knowledge. In fact, knowledge acquisition is a major bottleneck in RTE (Dagan et al., 2013, p. 7).

A successful RTE system can be leveraged in several other NLP tasks. For instance, in open-domain QA, an RTE system can rerank an output of a QA system (Harabagiu and Hickl, 2006). After the QA system returns a list of candidate answers to a question, e.g. "*How much fine does Barcelona have to pay?*", the RTE system can verify each candidate answer whether it entails a declarative version of the question where a question word acts as a variable, e.g., "*Barcelona has to pay X fine*". In Multi-document Summarization, a system is expected to generate a summary from several documents that are describing the same event. In this task, an RTE system can be used to detect redundant sentences in a summary, select the most responsive summary from possible candidates or model the semantic content of an ideal summary (Lacatusu et al., 2006). In Machine Translation, evaluation of a system translation against the reference (i.e. human) translation can benefit from an accurate RTE system (Pado et al., 2009). Instead of using string similarity measures for the evaluation, one can employ an RTE system to check whether the meanings of the system and reference translations are equivalent (i.e. entailing each other).

To sum up, NLI can be viewed as a soft version of logical entailment which is rooted in natural language. The RTE task is designed to assess NLI capabilities of NLP systems. The task represents "an abstract generic task that captures major semantic inference needs across applications" (Dagan et al., 2006). In the last decade, RTE datasets

[2]In the very first RTE dataset (Dagan et al., 2006), the canonical relations were two: ENTAILMENT and NON-ENTAILMENT. But since the fourth RTE challenge (RTE-4, Giampiccolo et al. (2008)) the non-entailment relation was further disambiguated as either CONTRADICTION or NEUTRAL.

[3]An RTE system is not expected to guess all gold labels in a dataset since even individual annotator's guesses do not completely match the gold labels. A performance ceiling for an RTE system is usually considered an average of all annotator performances.

are regularly created to exercise and test reasoning capacity of RTE systems in natural language. Moreover, since Mehdad et al. (2010) a cross-lingual version of the task is also being implemented. In the next section, we outline existing approaches to the RTE task.

1.2 Overview of approaches to textual entailment

There have been at least eight RTE challenges since the first RTE challenge (Dagan et al., 2006) and many diverse approaches have been suggested to solve the problem of RTE. Some of them favor rule-based methods operating on semantic representations and some of them prefer machine learning techniques applied to certain features extracted from an RTE problem. There are also baseline approaches that employ a lexical overlap measure between a text and a hypothesis. Before briefly summarizing existing approaches to an RTE task, let us first discuss an intuitive approach to the task.

Given an RTE problem $\langle T, H \rangle$, an *intuitive model* of an RTE system is expected first to express the meanings of T and H in some canonical semantic representation language and then to employ some sort of inference procedure to reason over the semantic representations (Dagan et al., 2013, Sec. 2.1). It is assumed that the inference procedure of the intuitive model is transparent and explanatory to a large extent rather than a black box with an opaque decision procedure. In general, approaches that follow the intuitive model are rule-based. It is assumed that rule-based systems are difficult to scale up since they require long-lasting elaborated work on several directions: knowledge acquisition, systematic translation of natural language texts into some meaning representation and automated reasoning for the meaning representation. Due to the difficulties related to the construction of an intuitive RTE model, the NLP community has sought alternative solutions to RTE. Below, we give a general overview of existing approaches and mention a few prominent RTE systems based on them.

Every approach to RTE employs some sort of representation for natural language text and an inference procedure over this representation. A representation of a linguistic expression can be shallow, like surface forms (i.e. a string representation) or a bag (i.e. a multiset) of its words, or relatively deeper, like its syntactic tree or a formula in some formal logic expressing its semantics. On the other hand, an inference procedure may vary from simple methods, based on set operations or an alignment (i.e. mapping phrases from T to H), to more elaborated methods, like reasoning with inference rules or employing some machine learning technique over certain features or objects derived from the representations. Several common representations and inference components are given in Figure 1.1. Inference procedures may vary in terms of the architecture. They can employ several components, e.g., a set of paraphrase rules with machine learning techniques, or a single component, e.g., an inference engine solely based on inference rules.

Depending on the depth of reasoning, approaches to RTE are conventionally considered as shallow, deep or somewhere in the middle. For instance, one of the extremely shallow approaches treats linguistic expressions as a bag of words and uses subset relation as entailment. According to it, (1) will correctly entail (4) as all words of the latter occurs in the former. But due to its simplicity, the approach cannot capture the entailment from (1) to (1a) because "*penalty*" does not occur in (1). Moreover, it makes severe mistakes like entailing (5) from (1).

Barcelona football club agreed to pay a €5.5m fine over the transfer of Brazil international Neymar in 2013 (1)

Barcelona agreed to pay a fine over the transfer (4)

Brazil agreed to pay a fine (5)

Following Adams (2006), the previous shallow method can be upgraded by considering surface forms and incorporating an alignment method, a word similarity measure and a machine learning classifier. A word similarity measure assigns a probability to each word pairs. The inference procedure works as follows. First, each content word of H is aligned (i.e. paired) with the most similar words in T. The product of the similarities of aligned words is considered as a similarity score of T and H. In this way, (4) and (5) are similar to (1) with a maximum score 1 taking into account that identical words are similar with a probability 1. Another feature used by the classifier is a number of negations, i.e. "*not*" and "*no*", modulo two. To indicate differences in surface forms, the number of gaps represents the number of content words in T that appear as a gap between the aligned words. For instance, when considering (1) and (4) as T and H respectively, the gaps are "*football*", "*club*" and "*€5.5m*". Feeding a trained decision tree classier with all these three features, i.e. a similarity score, the number of negations and the number of gaps, results in an RTE system with high performance. Surprisingly, only three RTE systems from 22 were able to outperform this shallow system at the second RTE challenge (Bar-Haim et al., 2006). Despite its high performance on the test dataset, the RTE system and its underlying approach is shallow since it can be easily misled. For instance, (4) and (5) obtain the same features, hence obtain the same entailment relation, with respect to (1).

There are also more elaborated versions of the previous approach. For classifiers, they usually employ more features extracted from different levels of representation. Lai and Hockenmaier (2014) offered one of such methods. Their system uses two representations at the same time: bag of words and chunked phrases combined with distributional and denotational semantics respectively. In distributional semantics, semantics of each word is modeled by the set of contexts it occurs in.[4] On the other hand, denotational semantics models a phrase based on the set of images that have the phrase in their caption. Based on these representations and semantics, they collect 10 features for each entailment problem with the help of various overlap, similarity and alignment techniques (see Figure 1.1). The collected features are then labeled with an entailment relation using a trained machine learning classifier. The resulted system achieves the highest score in the recent SemEval RTE task (Marelli et al., 2014a).[5] Despite its high performance, we place the approach relatively far from the intuitive model as it is not clear whether the approach

[4]More specifically, a word w is modeled in terms of an embedding vector $\vec{w}$, where each coordinate of $\vec{w}$ corresponds to a predefined context word. In the simplest case, the value of $\vec{w}_i$ is the number of occurrences of a word i in the contexts of w. The contexts are taken from some fixed large corpus. The distributional representation is motivated by the *distributional hypothesis* in linguistics saying that the words that occur in similar contexts tend to have similar meanings (Harris, 1955). For more details about distributional semantics and vector space semantics see (Turney and Pantel, 2010; Erk, 2012, and references therein).

[5]The task is intended for compositional distributional semantic approaches and contains relatively shorter and simpler problems than the original RTE challenges employ. The dataset of the task and the participant systems are further discussed in §6.1.1 and §6.4.2, respectively.

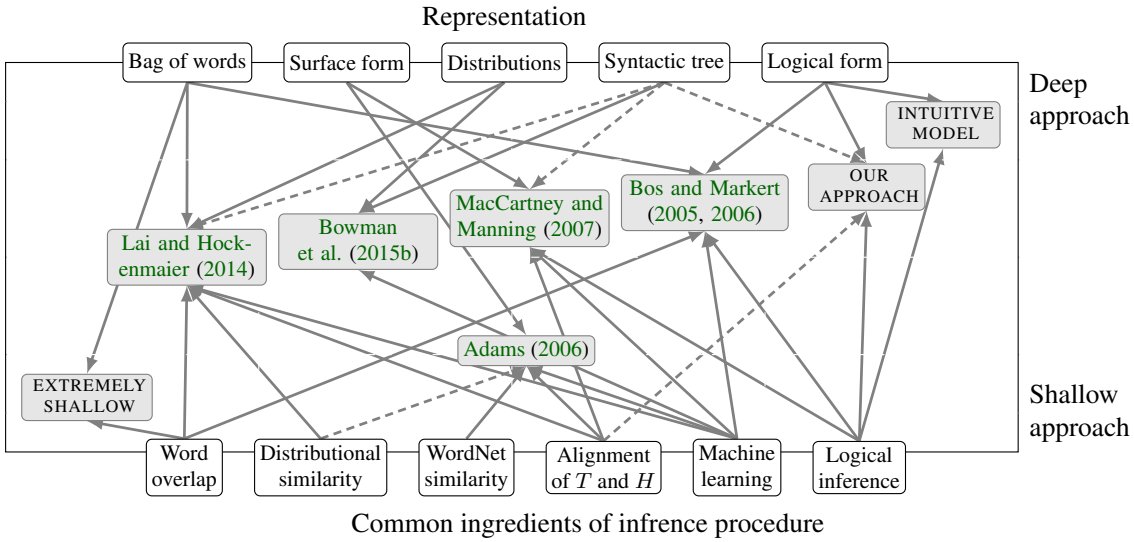


Figure 1.1: Rough display of some RTE systems and underlying approaches on the scale of the depth of NLI. Each system is linked with its representation and inference ingredients, where dashed arrows represent weak links. Knowledge resources are ignored for simplicity.

learns reasoning abilities or simple regularities found in the RTE dataset.[6]

Rule-based RTE systems with a set of inference rules and a transparent decision procedure come close to the intuitive model. Perhaps, this is the reason why around 40% of the systems at first RTE challenges were rule-based. But due to the difficulty to scale them up, the NLP community became less concerned about rule-based RTE systems. Nevertheless, some research lines still continue pursuing the rule-based paradigm. One of such lines is initiated by Bos and Markert (2005, 2006) who scaled up the idea of Blackburn and Bos (2005) to the RTE task. In particular, they translate T and H into first-order logic formulas (Bos et al., 2004) and employ off-the-shelf inference tools such as a theorem prover and a model builder to detect entailment relations. Unfortunately, Bos and Markert reported that the theorem prover correctly proves less than 6% of the RTE problems. A lack of lexical and background knowledge is considered as the main reason for it. On the other hand, their experiments showed that the combination of a machine learning classifier and the features extracted from the inference tools are more successful than the tools alone. With this decision their approach becomes more robust but shallow to some extent.[7]

Recent developments in compositional distributional semantics saw several new approaches to RTE that employ artificial neural networks for machine learning (Bowman et al., 2015b). Distributional representation of lexical semantics is found suitable for

[6]Moreover, on average, each fifth positive guess (i.e. ENTAILMENT or CONTRADICTION) of the RTE system is wrong. In other words, its precision is ca. 80%. More details about this issue is discussed in §6.4.2.

[7]Despite the usage of shallow techniques, we still put the approach high in Figure 1.1 as it employs elaborated logical forms with information on tense, events and thematic roles. On the other hand, the logical forms fall short of capturing higher-order semantics of quantifiers or adjectives, e.g., "*most*" and "*professional*", while our approach will properly account for it.

neural networks. To encode a word order or a tree structure of a linguistic expression, a special sentence encoder is included in the architecture of neural networks. A neural network approach is extremely greedy for labeled RTE datasets when learning entailment relations over linguistic expressions. Recently, an interest in this direction increased after Bowman et al. (2015a) made available a large RTE dataset with 570K sentence pairs. The primary objection to this kind of approaches is that they are not interpretable and their inference procedure is opaque.

Among other approaches to textual entailment, we would like to draw attention to the approach of MacCartney and Manning (2007, 2008, 2009), which is inspired by natural logic—a hypothetical logic that is built into natural language. As we will see later, our approach also builds on the idea of natural logic. MacCartney and Manning employed syntactic trees as a representation level where words are modeled by projectivity and implication signatures. Reasoning is done by transforming T into H using atomic edits. Each edit gives rise to one of seven elementary entailment relations. To obtain the final entailment relation, the elementary relations are composed according to the order of the edits. Due to its edit-driven reasoning, this approach can be considered as between shallow and deep methods. This approach is further discussed in §1.4.

As we have discussed, the approaches to textual entailment vary at least in terms of representation levels and inference procedures. Relatively shallow approaches are easy to scale up for wide-coverage text (e.g., there is no need for full semantic interpretation of natural language text) but can be easily deceived. In general, the systems based on shallow approaches are good in learning regularities from training RTE datasets. This makes them suitable for applications for open-domain text. On the other hand, deep approaches are much more reliable in their positive (i.e. ENTAILMENT or CONTRADICTION) answers but highly sensitive to errors in a semantic representation and to sparsity of knowledge and rules. It is usually difficult to train and scale up deep approaches. That is why, they are more used in applications with restricted domain. Despite these shortcomings, recently the interest in deep rule-based methods has been revived: combining logical semantics with distributional lexical semantics (Lewis and Steedman, 2013), and applying alignment techniques and knowledge acquisition on-the-fly (Tian et al., 2014). In §6.4, we compare our RTE system and the underlying approach to those mentioned here.

1.3 Natural logic and monotonicity

Natural logic is a hypothetical logic which is built in natural language and represents its integral part. Lakoff (1970), who coined the term of natural logic, considered it as a theory about "the regularities governing the notion of a valid argument for reasoning in natural language". Put differently, in natural language, natural logic characterizes valid arguments in the similar way as grammar does it for well-formed linguistic expressions. While doing so, natural logic operates on structures "as close as possible to the surface forms" (Moss, 2010b), unless surface forms per se. In this way, Aristotelian traditional logic, also called syllogistic logic, can be seen as a fragment of natural logic or vice versa, natural logic as a mature descendant of traditional logic. Since the thesis develops a theory inspired by natural logic, below we continue discussion on it by presenting monotonicity reasoning, i.e. a signature of natural logic.

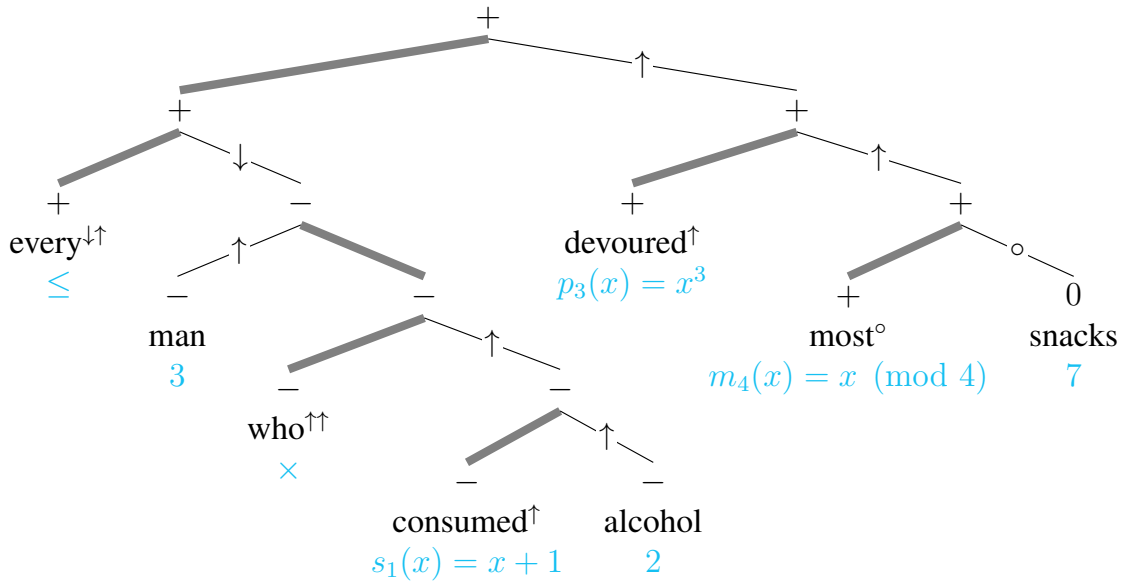


Figure 1.2: Underling semantic composition tree of the sentence in (6). The tree also encodes the syntactic tree of the first arithmetic expression from (8). Each function word is decorated with monotonicity properties. Function words are sitting on thick branches while arguments on thin ones. Each thin branch is marked with a monotonicity property projected by a function word. The root node has a positive polarity and polarities for the rest of the nodes are induced based on the monotonicity marks: the polarity of a node is the sign of the product of all monotonicity marks on the path from the node to the root (in a product, $\uparrow$, $\downarrow$ and $\circ$ are treated as $+1$, -1 and 0 respectively).

The most popular and success story of natural logic is *monotonicity reasoning*. It is a sort of reasoning with predicate substitution (including deletion and insertion). For example, consider the sentences in (6) and (7). One can obtain (7) from (6) by replacing "*man*", "*consumed*", "*alcohol*", "*devoured*" and "*most*" with "*young man*", "*drank*", "*beer*", "*ate*" and "*some*" respectively. While many sentences can be obtained from (6) by substitutions, only few of them are entailed from it. Monotonicity reasoning characterizes the substitutions that lead to entailments. This is done by determining a polarity (i.e. positive $+$, negative $-$ or neutral 0) for each constituent based on monotonicity properties of (function) words (i.e. upward monotonicity $\uparrow$, downward monotonicity $\downarrow$ and none of them $\circ$). Then the polarity of a constituent decides whether a certain substitution of the constituent yields an entailment relation.[8] Let us explain this procedure on the example.

Every man who consumed alcohol devoured most snacks (6)

Every^{+} man^{-} who^{-} consumed^{-} alcohol^{-} devoured^{+} most^{+} snacks^{0} (6a)

Every young man who drank beer ate some snacks (7)

$$3 \times s_1(2) \leq p_3(m_4(7)) \;\Rightarrow\; 2 \times s_1(1) \leq p_4(m_8(7)) \quad (8)$$

Words of certain classes, e.g., determiner, verb and conjunction, are rendered as functions. For instance, "*every*" is interpreted as a binary function: in (6), it takes "*man who*

[8] Notice that monotonicity is a lexical/local feature while polarity is contextual/global.

consumed alcohol" and "*devoured most snacks*" as arguments (see Figure 1.2). Each function word has monotonicity properties associated with its argument positions. The first argument position of "*every*" is downward monotone while the second one is upward monotone, written as "*every*$^{\downarrow\uparrow}$". A monotonicity property is understood in the similar way as in arithmetic. Informally speaking, "*every*" has similar monotonicity properties as the less or equal relation $\leq$, which itself can be seen as a binary function from numbers to truth values.

Monotonicity properties of function words project polarities on the arguments. As a result, each constituent in an expression gets a polarity depending on the context it occurs in. The polarities of constituents can be determined according to the following algorithm. An entire expression, i.e., a root node, gets the $+$ polarity. The polarity of a proper constituent, i.e. non-root nodes, is defined as follows. First, compute the product of all monotonicity properties on the path from the node of the constituent to the root (treat $\uparrow$, $\downarrow$ and $\circ$ as $+1$, -1 and 0 respectively), and then take the sign of the product (see Figure 1.2). The version of (6) with polarity marking on the words is given in (6a). We say that a substitution $A \rightsquigarrow B$ *obeys the polarity* $+$ (or $-$) if and only if A is semantically more specific (or general, respectively) than B.

Finally, we have the following result. If all applied substitutions obey the polarities of constituents in a source expression, the obtained expression is entailed from the source. As an example, inspect the entailment relation from (6) to (7). Notice that monotonicity reasoning can be carried out in the similar fashion on arithmetic expressions.[9] For instance, the reasoning process that validates entailment of (7) from (6) is isomorphic to the one that validates the entailment in (8).[10] See also Figure 1.2 for the polarity markings of the first arithmetic expression in (8).

A study of monotonicity reasoning as a formal calculus was initiated by van Benthem (1986, 1987) and further elaborated in Sánchez-Valencia (1991). As a part of the natural logic program, monotonicity calculus is often conjoined with a categorial grammar, e.g., Lambek Grammar (Lambek, 1958). Some works have focused on formal calculi for polarity marking, with an additional algorithm for marking (Sánchez-Valencia, 1991) or with internalized marking (Dowty, 1994; Moss, 2012). Fyodorov et al. (2003) developed and Zamansky et al. (2006) extended an inference system for a small fragment of English, which is based on monotonicity properties and directly operates on categorical grammar derivation trees. Research presented in Moss (2010a) and references therein departed from syllogistic logic and moved towards natural logic by extending the formal system with certain linguistic constructions and class of words. Muskens (2010) introduced a tableau proof system for a fragment of natural logic, formulas of which are typed λ-terms. Following MacCartney and Manning (2008), Icard (2012) presented a formal system for monotonicity reasoning extended with additional relations, e.g., exclusion and exhaustion. Finally, for a summary of the work on (extended) monotonicity reasoning and polarity marking see Icard and Moss (2014).

[9] To be on the safe side, we restrict the expressions to positive real numbers and assume the standard linear (pointwise) order over them when talking about the general/specific ordering relation.

[10] The function symbols $s_i(x)$, $p_i(x)$ and $m_i(x)$ denote $x + 1$, x^i and $x \pmod i$ respectively.

1.4 Natural logic approach to textual entailment

Application of natural logic for modeling NLI seems quite intuitive. After all it is conjectured to be logic native to natural language. This section contrasts a proof-based paradigm of natural logic to a semantic-based paradigm of translational approaches—the approaches that try to translate natural language expressions into an intermediate representation. Next, we describe the first mature natural logic-based approach (MacCartney and Manning, 2007, 2008, 2009) to textual entailment and highlight its shortcomings. In the end, we outline the motivation of our natural logic-based approach, which will be presented in the subsequent chapters of the thesis.

One of the main attractions of the natural logic program is that it attempts to describe "basic patterns of human reasoning directly in natural language without the intermediate of some formal system" (van Benthem, 2008b). While logical forms of natural logic are easily obtainable, this in general requires more work on the inferential part of the logic as the latter has to establish connection between superficial structures. On the other hand, in a translational approach, translation of linguistic semantics into some formal representation is usually much harder than developing the inferential part for it. This is because usually the translation already unfolds semantics of expressions to a large extent.

The mentioned contrast between natural logic and a translational approach brings us naturally to the distinction between *proof-theoretic* and *model-theoretic* (i.e. *truth-conditional*) semantics.[11] In particular, the natural logic program opts for proof-theoretic semantics as it models semantics of linguistic expressions in terms of proofs over the expressions. To the contrary, a translational approach adopts model-theoretic semantics: linguistic semantics are subject to truth with respect to a given model, i.e. a situation. Due to the relativity character of proof-theoretic semantics, certain types of reasoning comes easy to natural logic. For instance, from a natural logic perspective, entailing (4) from (1) simply requires *discarding* the qualifiers from (1) while a transnational approach first translates both sentences into a formal representation and then reasons over them. We believe that natural logic, while maintaining (most part of) surface forms, has a potential for more robust and economic reasoning in natural language.

Natural logic for an RTE task was first put to work by MacCartney and Manning (2007). Their applied fragment of natural logic included monotonicity calculus. Based on it, an RTE system, called NatLog, was evaluated against a small RTE dataset, a part of FraCaS (Cooper et al., 1996), and was also used as a component of a hybrid RTE system. Later, MacCartney and Manning (2008, 2009) augmented monotonicity calculus with additional semantic relations and implicative properties. Below we explain their natural logic approach to RTE by using an example in Figure 1.3, borrowed from MacCartney (2009) and slightly modified.

In their version of natural logic, MacCartney and Manning employed seven entailment relations; some of them are: equivalence (smart $\equiv$ clever), forward-entailment (dance $\sqsubset$ move), backward-entailment (move $\sqsupset$ dance), alternation (cat $|$ dog) and negation (human

[11] The goal of model-theoretic semantics is to characterize meaning as a condition that explains in which situations the meaning is true and in which false. On the other hand, proof-theoretic semantics aims to model meaning with respect to other meanings, i.e. in terms of the entailment relations it has with other meanings, where the relations are verified by proofs.

	Sentence & Atomic edit	Lexical	Projected	Overall
(S0)	John refused to move without blue jeans			
	(E1) DEL(refused to)	|	|	|
(S1)	John moved without blue jeans			
	(E2) INS(didn't)	^	^	⊏
(S2)	John didn't moved without blue jeans			
	(E3) SUB(move, dance)	⊐	⊏	⊏
(S3)	John didn't dance without blue jeans			
	(E4) DEL(blue)	⊏	⊏	⊏
(S4)	John didn't dance without jeans			
	(E5) SUB(jeans, pants)	⊏	⊏	⊏
(S5)	John didn't dance without pants			

Figure 1.3: An example of entailing (S5) from (S0) in the natural logic of MacCartney and Manning (2008). A lexical relation is solely determined by an atomic edit. A projected relation depends on a lexical relation and a polarity of an edited site. An overall relation, with respect to the initial phrase (S0), is a composition of projected relations.

^ nonhuman).[12] In order to find out an entailment relation between T and H—in our example, (S0) and (S5) respectively—a sequence of atomic edits are found that transforms T into H. For instance, (S0) is transformed into (S5) via the sequence (E1-E5); see Figure 1.3. For each atomic edit, a corresponding lexical entailment relation is determined. For example, we have *refuse to* $P \mid P$ based on the implicative signature of "*refused to*". Hence, its deletion (E1) corresponds to ($\mid$) lexical relation.

Next, the lexical entailment relations are projected to the sentence level. Ideally, these projections follow the semantic composition tree (like the one in Figure 1.2) of the edited expression, but MacCartney and Manning's system NatLog carries out the projection based on phrase-structure trees, where predefined tree patterns for monotonic entries are employed to determine polarities.[13] A projected relation represents an entailment relation between two sentences differing from each other by a single atomic edit. As a result, (S0) | (S1) holds after projecting the lexical entailment relation triggered by the edit (E1). To illustrate an effect of negative polarity, consider the edit (E3). The lexical relation ($\sqsupset$) of (E3) is reversed after the projection since the edition occurs in a negative polarity context, under the scope of "*didn't*". After all lexical entailment relations are projected, we get a chain of entailment relations and intermediate expressions from T to H. A composition of the projected relations yields the final entailment relation ($\sqsubset$) from T to H:

$$(S0) \mid (S1) \bowtie (S1)\text{^}(S2) \bowtie (S2)\sqsubset(S3) \bowtie (S3)\sqsubset(S4) \bowtie (S4)\sqsubset(S5) = (S0)\sqsubset(S5) \quad (9)$$

Despite the elegance and simplicity of the described approach, it has one major drawback. The reasoning in the approach is guided by a sequence of atomic edits, i.e. alignment of T to H. This guidance significantly limits the reasoning capacity of the framework. For example, MacCartney (2009) notes that not all alignments of T to H lead to

[12] The alternation and negation relations can be seen as a specification of an exclusion relation.

[13] More details about the entailment projection in NatLog and its shortcomings see (MacCartney, 2009, Sec. 7.5).

the same final entailment relation. Moreover, it is not clear how to search the alignments that yield a correct and specified relation, if there is such a relation.[14]

The alignment-driven reasoning has also its limitations in terms of coverage. For instance, MacCartney (2009) observes that his framework cannot account for de Morgan's laws for quantifiers, one of them exemplified by PROB-PROB-1. Furthermore, it is not possible to account for the entailment relation in PROB-PROB-2, encoding one of de Morgan's laws for Booleans. The entailments licensed by an alternation of a structure, see PROB-PROB-3 and PROB-4, are also beyond competence of the framework. Bound to alignment of two phrases, the approach falls short of reasoning over several premises, e.g. PROB-PROB-5. For the latter reason, the NatLog system was only evaluated against single-premised problems in FraCaS. After all, we can conclude that the natural logic approach a la MacCartney and Manning, though simple and crafty, is significantly crippled by the usage of the alignment technique which has no connection with natural logic.

Not all bird fly	
Some birds does not fly	
PROB-1	

Mary is not blond and is not tall	
Mary is not blond or tall	
PROB-2	

John bought a car from Bill	
Bill sold a car to John	
PROB-3	

A student wrote an essay	
An essay was written by a student	
PROB-4	

Most children like candies Most children hate porridge	
Some children like candies and hate porridge	
PROB-5	

Apart from the above described implementation of natural logic, there has been a little work on building computational models for reasoning based on natural logic. Fyodorov et al. (2003) implemented monotonicity reasoning over categorial grammar derivation trees which allows coordination and relative clauses. Eijck (2005) gave a preliminary implementation of syllogistic logic with monotonicity and symmetry rules. Hemann et al. (2015) discussed two implementations of extended syllogistic logics. All these implementations are restricted to a small fragment of natural language.

The objective of the current thesis is to account for reasoning in natural language by employing a version of natural logic. While doing so, we aim to fill the gap that remains between studies of formal inference systems for natural logic and their application to wide-coverage textual entailment. In this way, our approach will deliver a theory and a computational model for wide-coverage natural reasoning, where employed logical forms resemble surface forms, and at the same time, we maintain *healthy* logical reasoning over these forms. To achieve this goal, we base on the novel idea of Muskens (2010) to device a semantic tableau method for natural logic, referred here as a *natural tableau*. Muskens opts for a version of natural logic that represents a higher-order logic with simply typed λ-terms, where lexical terms are the only constant terms. Logical forms come close to the semantic composition trees which were used for detecting polarities in Figure 1.2.

Reasoning over the terms is carried out according to a specially designed semantic tableau method. Basically, the tableau method represents a collection of inference rules

[14] The framework allows underspecified entailment relations too. For instance, $\{\sqsubset, |, \frown\}$ is an underspecified entailment relation that might be one of the three specified entailment relations.

that unfold semantics of terms by breaking them into smaller constituent pieces. The rules are used to unfold the semantics of terms and find out inconsistency between them. Thus, the reasoning procedure based on a tableau method has a nature of refutation. For instance, entailment of H from T is proved by finding T and negation of H semantically inconsistent. Unlike the approach of MacCartney and Manning (2007), the reasoning via a semantic tableau is not limited to single-premised arguments. Moreover, accounting for monotonicity reasoning, Booleans and sentence alternations will not represent a problem for our approach.

1.5 Overview of what follows

The section gives an overview of the rest of the chapters. The chapters are organized in a self-contained way. Each chapter starts with a brief outline. If some prior knowledge is required for comprehension of the material, the chapter starts with a preliminary section.

Chapter 2 starts with preliminaries concerning a functional type theory and a semantic tableau method. Familiarity with these two theories is important for understanding formal theory behind the analytic tableau method of Muskens (2010), which is introduced next. The rest of the chapter describes extension of the analytic tableau method in three directions. First, we describe the extension of a type system with syntactic types. The latter can be seen as an integration of syntactic information in logical forms and reasoning. This step will make it easy to keep logical forms similar to linguistic expressions, and syntactic information will be used to unfold semantics of various linguistic constructions accordingly. Second, the format of tableau entries is extended with an additional slot which serves as a storage for remote (i.e. indirect) modifiers. The storage makes it easy to account for event semantics in the tableau system. Last, a set of new tableau rules is introduced that concerns interaction between modifiers and the new slot and accounts for the semantic exclusion and exhaustion relations. The three-fold extension gears the tableau system for wide-coverage natural reasoning. The chapter extends the work presented by Abzianidze (2015c).

Chapter 3 takes a break from the discussion of the tableau method and describes a way to obtain the logical forms of natural logic automatically from Combinatory Categorial Grammar (CCG) derivation trees. The procedure consists of several components: (i) getting CCG terms by removing a directionality from CCG derivation trees, (ii) correcting CCG terms by eliminating several systematic mistakes made by CCG parsers, and (iii) obtaining a final logical form, called Lambda Logical Forms (LLFs), by type-raising quantified noun phrases in corrected CCG terms. All in all, the procedure takes a CCG derivation tree and outputs a list of logical forms (see Figure 1.4). The number of logical forms is conditioned by quantifier scope ambiguity. The designed LLF generator (LLFgen) can be used for many semantic applications as it constructs structures similar to the semantic compositional trees (see Figure 1.2). The generation of LLFs was shortly presented by Abzianidze (2015c,b).

Chapter 4 continues discussion of the tableau method and presents a wide-range of tableau rules necessary for reasoning over open-domain text. After knowing from Chapter 3 how the logical forms for wide-coverage sentences looks like, it is easier to present these rules. The chapter covers tableau rules for adjectives, prepositional phrases, definite determiners, expletives, verb subcategorization, open compound nouns, light verb con-

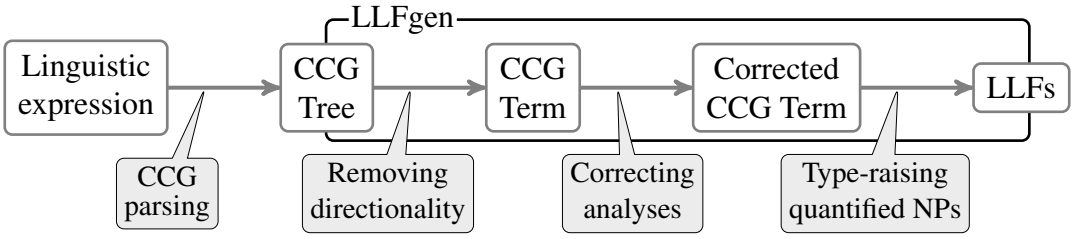


Figure 1.4: The LLF generator producing LLFs from a CCG derivation tree.

structions, copula, passive constructions, attitude verbs, etc. Most of the rules presented in the chapter are refined versions of the rules found in Abzianidze (2015c).

Chapter 5 describes the architecture of a tableau theorem prover for natural language, called LangPro. First, the issues of natural language theorem proving, such as knowledge extraction from WordNet and strategies for rule applications in the tableau system, are discussed. Then functionality of two theorem provers are described. One is a natural logic theorem prover (NLogPro), which operates on logical forms, while another prover (LangPro) reasons over natural language expressions. The latter prover contains a CCG parser, LLFgen and an aligner along with NLogPro (see Figure 1.5). The aligner aligns shared sub-terms of LLFs in order to prevent NLogPro from unnecessary rule applications.

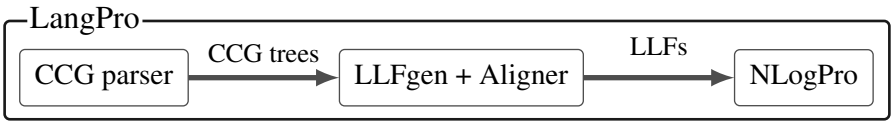


Figure 1.5: The architecture of a natural language theorem prover, called LangPro.

Chapter 6 discusses learning and evaluation phases for the theorem prover on the RTE datasets SICK (Marelli et al., 2014b) and FraCaS (Cooper et al., 1996). The learning phase consists of adaptation and development. The former involves collecting tableau rules, enriching the knowledge base and designing fixing rules for CCG terms. In other words, the tableau rules from Chapter 4 and the correction rules from Chapter 3 can be seen as deliverables of the adaptation phase. The development phase, on the other hand, is used to set optimal parameters for the theorem prover. Due to a small size of FraCaS, we use the same FraCaS sections for learning and evaluation. For SICK, different portions are used for adaptation, development and evaluation. The evaluation reveals that the prover is extremely reliable in its proofs, and obtains competitive results on each dataset. On the FraCaS dataset, LangPro demonstrates state-of-the-art competence while on SICK it achieves performance close to average human performance. The obtained results are compared to related RTE systems. The work concerning the SICK and FraCaS datasets were presented by Abzianidze (2015b,a) and Abzianidze (2016), respectively.

Chapter 7 concludes the thesis and discusses future work on the natural tableau system. In particular, we present future research concerning the RTE datasets of different type, lexical knowledge acquisition, coupling with distributional semantics, and generating LLFs from dependency trees.

Chapter 2

Natural Tableau for Natural Reasoning

The chapter presents a tableau system that operates on logical forms of linguistic expressions. We start with the introduction of a type theory that is used throughout the work as a semantic representation. For the readers who are not familiar with a semantic tableau method, we introduce the idea behind the method and present a propositional tableau system. After the preliminaries, the tableau system for natural logic of Muskens (2010) is introduced. First, the Lambda Logical Form (LLF) and the format of tableau entries are discussed, and then tableau rules with demonstrative proofs are presented. In order to make the system suitable for reasoning over unrestricted natural language text, we extend it in several directions. The first direction is the formal language. In particular, we propose to extend the type system by adding syntactic types and adopt a subtyping relation over the types in order to establish interaction between the syntactic and semantics types. The next extension is directed towards the format of tableau entries. A new slot in a tableau entry serves as a memory that keeps modifiers. This change enables smooth integration of event semantics in the tableau system without altering the logical forms. The final step in the development of the tableau system is to enhance the inventory of rules. We start with presenting the rules specially designed for the newly added slot. Along with these rules, algebraic properties relevant to reasoning are also introduced. We augment the reasoning power of the system by modeling the exclusion and exhaustion relations. In particular, we give the rules for these relations concerning lexical entries. Then the rules modeling the interaction of these relations and monotone operators are presented. The rules that account for several projectivity properties of functions are also introduced.

2.0 Preliminaries

The section introduces two important formal devices, a type theory (Church, 1940; Henkin, 1950) and a tableau method (Beth, 1955; Hintikka, 1955) which are employed throughout the thesis. A simple type theory will be presented in a general way, and it will correspond to a family of type logics differing in terms of a type system. The theory serves as a formal language for representing linguistic semantics. We give the syntax and semantics of its terms with several abbreviations and writing conventions. A tableau method represents a proof procedure that proves theorems by attempting to refute them. We present the propositional version of a tableau method by describing the intuition behind it and exemplifying a tableau proof tree with tableau rules. The combination of these two devices,

the tableau system for a type theory, is a topic of discussion in the subsequent sections.

2.0.1 Functional type theory

The subsection presents a simple type theory, i.e. type logic, with the interpretations in terms of functions. We will present a family of type logics which will be refered as TY after Gallin (1975). The TY logic represents a version of higher-order logic and is used throughout the work as a semantic representation language for linguistic expressions. We start with introducing the formal syntax of TY language, which basically follows Church (1940), and then give standard functional semantics for its terms.

Definition 1 (TY type system)**.** A TY *type system* over a non-empty set of *basic types*, denoted by $\mathscr{T}_{\mathcal{B}}$, is the smallest set of types such that:

(i) $t \in \mathcal{B}$, i.e. a *truth type* is a basic type;

(ii) $\mathcal{B} \subset \mathscr{T}_{\mathcal{B}}$, i.e. all basic types are types;

(iii) If $\alpha \in \mathscr{T}_{\mathcal{B}}$ and $\beta \in \mathscr{T}_{\mathcal{B}}$, then $(\alpha, \beta) \in \mathscr{T}_{\mathcal{B}}$, where (α, β) is a *function type*.

A common TY type system is $\mathscr{T}_{\mathtt{Sem}}$ where the set of basic types consists of an entity type e and a truth type t, i.e. $\mathtt{Sem} = \{e, t\}$. Type logic $\mathtt{TY}_1$, which employs the type system $\mathscr{T}_{\mathtt{Sem}}$, will be used in §2.1 when introducing a tableau system.

The Greek letters α, β, γ will be used as meta-variables ranging over the TY types. The function types sometimes will be abbreviated by omitting (outer) parentheses with the convention that association is to the right; commas will also be ignored in case of the basic types with single letters. For instance, eet stands for $(e, (e, t))$. Intuitively, $(\alpha\beta)$ stands for the type of unary function which takes an argument of type α and returns a value of type β. In this way, $(\mathbb{R}, \mathbb{N})$ would be the type of functions from the real numbers $\mathbb{R}$ to the natural numbers $\mathbb{N}$.

The formal language of TY represents the simply typed λ-calculus (Church, 1940) based on the TY types. We assume that for each TY type α we have countably infinite constants and variables of that type, denoted by the sets $\mathcal{C}_\alpha$ and $\mathcal{V}_\alpha$ respectively. By default, the trailing letters of the alphabet x, y, z and their indexed versions $x^1, ..., z^n$ are reserved for variables. The terms of TY are defined in a recursive way:

Definition 2 (TY terms)**.** For each TY type α, a set of TY *terms* of type α, denoted by $\mathcal{T}_\alpha$, is the smallest set of terms such that:

(i) $(\mathcal{C}_\alpha \cup \mathcal{V}_\alpha) \subset \mathcal{T}_\alpha$; *Basic terms*

(ii) If $B \in \mathcal{T}_{\alpha\beta}$ and $A \in \mathcal{T}_\alpha$, then $(BA) \in \mathcal{T}_\beta$; *Function application*

(iii) If $x \in \mathcal{V}_\alpha$ and $B \in \mathcal{T}_\beta$, then $(\lambda x B) \in \mathcal{T}_{\alpha\beta}$; *$\lambda$-abstraction*

(iv) If $A, B \in \mathcal{T}_t$, i.e. A and B are *formulas*, then $\neg A \in \mathcal{T}_t$ and $(A \wedge B) \in \mathcal{T}_t$; *Negation & Conjunction*

(v) If $A \in \mathcal{T}_t$ and $x \in \mathcal{V}_\alpha$, then $\forall x A \in \mathcal{T}_t$; *Universal quantification*

(vi) If $A, B \in \mathcal{T}_\alpha$, then $(A = B) \in \mathcal{T}_t$. *Equality*

We ignore (outer) parentheses in the terms where it can be done without causing ambiguity. While doing so, we assume association to the left and the widest scope for λ-abstraction and universal quantification. Sometimes for better readability we employ a

dot after a variable in λ-abstraction or universal quantification. For instance, we write $\forall y.AB(\lambda x.\, xC)$ instead of $\forall y((AB)(\lambda x(xC)))$.

In order to present the type of a term in a compact way, we put the type in the subscript, e.g., A_α implies that $A \in \mathcal{T}_\alpha$. Also the colon format $A : \alpha$ will be used to indicate the type of a term. Sometimes the types are ignored for the variables in universal quantification and λ-abstraction, e.g., $\forall x(x_\beta A_\alpha)$ is the same as $\forall x_\beta(x_\beta A_\alpha)$. In Definition 2, the terms A and B are *sub-terms* of the *compound* terms constructed in (ii)–(vi). Notice that the definition also encodes *type inference rules*—the rules that determine the type of a compound term based on the types of its sub-terms.

Definition 3 (Free and bound variables)**.** An occurrence of a variable x in a term A is *bound* if it occurs in a sub-term of A of the form (λxB) or $\forall xB$; otherwise the occurrence is *free*. A term with no free occurrence of a variable is called *closed*; otherwise it is *open*.

For example, $AB(\lambda x.\, xC)$ and $A\forall x(BxC)$ are closed while $(\lambda x.AyB)$ and $\forall x(AxyB)$ are open as y occurs freely there.

After defining the TY terms and types, we need to give semantics for the terms. Each term will be interpreted as a function from functions to functions, where every function under discussion takes at most one argument. According to this functional interpretation, the arithmetic operation of addition $(+)$ is understood as a unary function from real numbers, i.e. functions with no arguments, to functions that map real numbers to real numbers. This trick of expressing functions of several arguments in terms of unary functions is due to Schönfinkel. For the interpretation we need a collection of functions with domains and codomains of various types.

Definition 4 (TY frame)**.** A TY frame is a set $\mathcal{F}_\mathcal{B} = \{\mathcal{D}_\alpha \mid \alpha \in \mathcal{T}_\mathcal{B}\}$ such that for any α and β types:

(i) $\mathcal{D}_\alpha \neq \emptyset$;

(ii) $\mathcal{D}_t = \{0, 1\}$ is a standard Boolean algebra on $\{0, 1\}$;

(iii) $\mathcal{D}_{\alpha\beta} \subseteq \mathcal{D}_\beta^{\mathcal{D}_\alpha}$, where the latter is a set of all functions from $\mathcal{D}_\alpha$ into $\mathcal{D}_\beta$.

A TY frame is *standard* if $\mathcal{D}_{\alpha\beta} = \mathcal{D}_\beta^{\mathcal{D}_\alpha}$ in (iii). Semantics of TY logic will be defined in terms of standard frames.

In order to interpret TY terms on a standard TY frame, we first interpret basic terms and then on their basis compound terms are interpreted. An *interpretation function* $\mathcal{I}$ for a TY frame $\mathcal{F}$ is a function from a set of constant terms $\mathcal{C}_\alpha$ to $\mathcal{D}_\alpha$. An *assignment function* $\mathfrak{a}$ is a function from a set of variables $\mathcal{V}_\alpha$ to $\mathcal{D}_\alpha$. By $\mathfrak{a}[b/x]$ we denote an assignment function such that $\mathfrak{a}[b/x](x) = b$ and $\mathfrak{a}[b/x](y) = \mathfrak{a}(y)$ if $x \neq y$. In other words, $\mathfrak{a}[b/x]$ coincides with $\mathfrak{a}$ but it maps x to b. A *standard model* is a pair $\mathcal{M} = \langle \mathcal{F}, \mathcal{I} \rangle$, where $\mathcal{I}$ is an interpretation function for a standard TY frame $\mathcal{F}$.

Definition 5 (Standard semantics of TY)**.** The *standard semantics* of a TY term A with respect to a standard model $\mathcal{M} = \langle \mathcal{F}, \mathcal{I} \rangle$ and an assignment $\mathfrak{a}$ is denoted by $[\![A]\!]^{\mathcal{M},\mathfrak{a}}$ and is defined as:[1]

[1] For *generalized semantics* of TY with respect to a *general model* (Henkin, 1950), i.e. a model based on a general frame, existence of such $[\![.]\!]^{\mathcal{M},\mathfrak{a}}$ semantic function is required while for a standard model $[\![.]\!]^{\mathcal{M},\mathfrak{a}}$ exists naturally.

(i) $[\![c]\!]^{\mathcal{M},\mathfrak{a}} = \mathcal{I}(c)$ for any constant term c and
$[\![x]\!]^{\mathcal{M},\mathfrak{a}} = \mathfrak{a}(x)$ for any variable term x;

(ii) $[\![AB]\!]^{\mathcal{M},\mathfrak{a}} = [\![A]\!]^{\mathcal{M},\mathfrak{a}}([\![B]\!]^{\mathcal{M},\mathfrak{a}})$;

(iii) $[\![\lambda x_\beta\, A_\alpha]\!]^{\mathcal{M},\mathfrak{a}} = f \in \mathcal{D}_\alpha^{\mathcal{D}_\beta}$ such that for any $b \in \mathcal{D}_\beta$, $f(b) = [\![A_\alpha]\!]^{\mathcal{M},\mathfrak{a}[b/x]}$;

(iv) $[\![\neg A_t]\!]^{\mathcal{M},\mathfrak{a}} = 1 - [\![A_t]\!]^{\mathcal{M},\mathfrak{a}}$, i.e. a Boolean complement;
$[\![A_t \wedge B_t]\!]^{\mathcal{M},\mathfrak{a}} = \inf\{[\![A_t]\!]^{\mathcal{M},\mathfrak{a}}, [\![B_t]\!]^{\mathcal{M},\mathfrak{a}}\}$, where infimum is taken from a Boolean algebra on $\{0,1\}$;

(v) $[\![\forall x_\alpha A_t]\!]^{\mathcal{M},\mathfrak{a}} = \inf\limits_{b\in\mathcal{D}_\alpha}\{[\![A_t]\!]^{\mathcal{M},\mathfrak{a}[b/x]}\}$.

(vi) $[\![A_\alpha = B_\alpha]\!]^{\mathcal{M},\mathfrak{a}} = 1$ if $[\![A_\alpha]\!]^{\mathcal{M},\mathfrak{a}} = [\![B_\alpha]\!]^{\mathcal{M},\mathfrak{a}}$; otherwise $[\![A_\alpha = B_\alpha]\!]^{\mathcal{M},\mathfrak{a}} = 0$

If A is a closed term, its semantics does not depend on an assignment function. So, we simply write $[\![A]\!]^{\mathcal{M}}$ instead of $[\![A]\!]^{\mathcal{M},\mathfrak{a}}$. It can be checked that the semantic interpretation of a term does not change while the term undergoes the standard λ-conversions: α-conversion, β-conversion and η-conversion. For example, A is obtained from $(\lambda x_\beta.\, Ax_\beta)$ after η-conversion when x has no free occurrence in A. To show that both terms have the same semantics, let us see how the semantics of $(\lambda x_\beta.\, Ax_\beta)$ behaves. For any standard model $\mathcal{M}$, any assignment function $\mathfrak{a}$ and any $b \in \mathcal{D}_\beta$, we have:

$$[\![\lambda x.\, Ax]\!]^{\mathcal{M},\mathfrak{a}}(b) \overset{\text{(iii)}}{=} [\![Ax]\!]^{\mathcal{M},\mathfrak{a}[b/x]} \overset{\text{(ii)}}{=} [\![A]\!]^{\mathcal{M},\mathfrak{a}[b/x]}([\![x]\!]^{\mathcal{M},\mathfrak{a}[b/x]}) \overset{\text{(i)}}{=} [\![A]\!]^{\mathcal{M},\mathfrak{a}[b/x]}(b)$$

By the assumption that x has no free occurrence in A, we have $[\![A]\!]^{\mathcal{M},\mathfrak{a}[b/x]} = [\![A]\!]^{\mathcal{M},\mathfrak{a}}$. The latter means that $[\![A]\!]^{\mathcal{M},\mathfrak{a}}(b) = [\![\lambda x.\, Ax]\!]^{\mathcal{M},\mathfrak{a}}(b)$ for any $b \in \mathcal{D}_\beta$. Hence, the two interpretations are the same functions (according to the axiom of extensionality of Zermelo-Fraenkel set theory).

The semantics of the terms $\neg A_t$, $A_t \wedge B_t$ and $\forall x A_t$ are classical. Based on them we can define the terms with other classical operators.

Definition 6 (Logical operators)**.** The following terms of type t are defined as:

(a) $A_t \vee B_t \overset{\text{def}}{=} \neg(\neg A_t \wedge \neg B_t)$; *Disjunction*

(b) $A_t \to B_t \overset{\text{def}}{=} \neg(A_t \wedge \neg B_t)$;[2] *(Material) Implication*

(c) $\exists x_\alpha A_t \overset{\text{def}}{=} \neg\forall x_\alpha \neg A_t$; *Existential quantification*

To allow writing certain long terms in a compact way, we introduce several conventions. If a term is formed by several λ-abstractions in a row, then we write a single λ followed by a sequence of variables, e.g., we write $\lambda xyz.\, A$ instead of $\lambda x \lambda y \lambda z.\, A$. We will use a *vector representation* for sequences of terms. For instance, $\lambda x^1 x^2 \ldots x^n . A$ will be abbreviated as $\lambda \vec{x} . A$, where $\vec{x} = x^1, ..., n^n$ for some natural number n. By default, we assume that variables in variable vectors or sequences are different from each other. The same conventions work for several universal or existential quantifications, e.g., we write $\forall \vec{x} . A$ instead of $\forall x^1 \forall x^2 ... \forall x^n . A$. Using the vector representation we abbreviate the terms of the form $AB^1 B^2 \ldots B^n$, where A is applied to a sequence of n terms, by $A\vec{B}$ where

[2] We do not define the material equivalence separately in terms of $(A_t \to B_t) \wedge (B_t \to A_t)$ as it already coincides with $A_t = B_t$.

$\vec{B} = B^1, \ldots, B^n$. Notice that $\vec{B}$ in $A\vec{B}$ is not a sub-term due to the left association of the function application. On the other hand $\vec{B}$ can be rendered as a sub-term in $\vec{B}C$, but we will not use the latter notation in order to avoid confusion. For the types, the vector representation is used to shorten the types of the form $\alpha_1\alpha_2 \ldots \alpha_n$ as a vector type $\vec{\alpha}$ where $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$. For example, $(et)(et)t$ can be denoted as $\vec{\alpha}t$, where $\vec{\alpha} = (et), (et)$. We say that a vector term $\vec{A} = A^1, \ldots, A^n$ is of vector type $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$ if A^i is of type α_i. The latter is written in short as $\vec{A}_{\vec{\alpha}}$. It is important to see the difference between two usages of vector types depending the kind of term it accompanies: $\vec{A}_{\vec{\alpha}}$ vs $A_{\vec{\alpha}}$. In the first example, $\vec{\alpha}$ is read as a sequence of types while in the second example it represents a single type.

The terms of type $\vec{\alpha}t$, where $\vec{\alpha}$ is any (possibly empty) sequence of the types, are of special interest. The semantic interpretation of such a term corresponds to an n-ary function with values in $\mathcal{D}_t = \{0, 1\}$, where n is a length of $\vec{\alpha}$. For instance, $[\![A_{eet}]\!]^{\mathcal{M},\mathfrak{a}}$ is a function from $\mathcal{D}_e$ to $\mathcal{D}_t^{\mathcal{D}_e}$, which corresponds to a binary function from $\mathcal{D}_e \times \mathcal{D}_e$ to $\mathcal{D}_t$ according to Schönfinkel's trick. Since the functions to $\{0, 1\}$ are characteristics for sets and relations, the semantics of terms of type $\vec{\alpha}t$ can also be interpreted as sets or relations. Due to this connection, the types of the form $\vec{\alpha}t$ are called as *relational types*, members of $\mathcal{D}_{\vec{\alpha}t}$ as *characteristic* or *relational functions*, and the terms of relational type as *relational terms*.

For each relational type, denoted as $\vec{\alpha}t$, the set $\mathcal{D}_{\vec{\alpha}t}$ of functions represents a Boolean algebra isomorphic to the Boolean algebra over the powerset $\wp(\mathcal{D}_{\alpha_1} \times \ldots \times \mathcal{D}_{\alpha_n})$ where $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$. The partial pointwise order over functions of $\mathcal{D}_{\vec{\alpha}t}$ is defined recursively. For any $f, g \in \mathcal{D}_{\vec{\alpha}t}$, we have $f \leq g$ iff $f(b) \leq g(b)$ for any $b \in \mathcal{D}_\beta$, where $\vec{\alpha} = \beta\vec{\gamma}$. Notice that the defined partial order is induced by $(\leq)$ of $\mathcal{D}_t = \{0, 1\}$. We denote the least and greatest elements of $\mathcal{D}_{\vec{\alpha}t}$ as $0_{\vec{\alpha}t}$ and $1_{\vec{\alpha}t}$ respectively; hence $0_t = 0$ and $1_t = 1$.

Relational terms are as flexible as their interpretations. In particular, it makes sense to have Boolean connectives for relational terms similarly to formulas.

Definition 7 (Generalized Boolean connectives)**.** The terms formed from relational terms via the generalized Boolean connectives are defined as follows:

(e) $-A_{\vec{\alpha}t} \stackrel{\text{def}}{=} \lambda\vec{x}.\, \neg(A_{\vec{\alpha}t}\vec{x}_{\vec{\alpha}})$; *Complement / gen. negation*

(f) $A_{\vec{\alpha}t} \sqcap B_{\vec{\alpha}t} \stackrel{\text{def}}{=} \lambda\vec{x}.\, A_{\vec{\alpha}t}\vec{x}_{\vec{\alpha}} \wedge B_{\vec{\alpha}t}\vec{x}_{\vec{\alpha}}$; *Meet / gen. conjunction*

(g) $A_{\vec{\alpha}t} \sqcup B_{\vec{\alpha}t} \stackrel{\text{def}}{=} \lambda\vec{x}.\, A_{\vec{\alpha}t}\vec{x}_{\vec{\alpha}} \vee B_{\vec{\alpha}t}\vec{x}_{\vec{\alpha}}$; *Join / gen. disjunction*

(h) $A_{\vec{\alpha}t} \sqsubseteq B_{\vec{\alpha}t} \stackrel{\text{def}}{=} \forall\vec{x}(A_{\vec{\alpha}t}\vec{x}_{\vec{\alpha}} \rightarrow B_{\vec{\alpha}t}\vec{x}_{\vec{\alpha}})$; *Subsumption / gen. implication*

The semantics of the defined terms are intuitive. For instance, the interpretation of $A_{\vec{\alpha}t} \sqcap B_{\vec{\alpha}t}$ is the relation which is an intersection of the relations corresponding to the interpretations of $A_{\vec{\alpha}t}$ and $B_{\vec{\alpha}t}$. The rest of the connectives are also interpreted in terms of the operations (e.g., $\cup$ and $-$) and the relation $\subseteq$ over sets. Subsumption over the terms of type $\vec{\alpha}t$ can be interpreted as the pointwise order $\leq$ over the functions in $\mathcal{D}_{\vec{\alpha}t}$. Generalized equivalence defined in terms of $(A_{\vec{\alpha}t} \sqsubseteq B_{\vec{\alpha}t}) \wedge (B_{\vec{\alpha}t} \sqsubseteq A_{\vec{\alpha}t})$ coincides with $A_{\vec{\alpha}t} = B_{\vec{\alpha}t}$. Notice that the defined terms, except the one in (h), are of type $\vec{\alpha}t$. When $\vec{\alpha}$ and $\vec{x}$ are empty sequences, then these connectives coincide with the classical propositional connectives. The usage of the classical connectives automatically hints that the terms are formulas.

The notions of validity and entailment over TY terms is defined in a standard way:

Definition 8 (Standard validity & entailment)**.** A formula A_t is *standardly valid* or *s-valid*, written as $\vDash A_t$, if $[\![A_t]\!]^{\mathcal{M},\mathfrak{a}} = 1$ for any standard model $\mathcal{M}$ and any assignment $\mathfrak{a}$. A finite set of formulas Γ *standardly entails* or *s-entails* a finite set of formulas Δ, written as $\Gamma \vDash \Delta$, if and only if for any standard model $\mathcal{M}$ and any assignment $\mathfrak{a}$:

$$\inf\{[\![A_t]\!]^{\mathcal{M},\mathfrak{a}} \mid A_t \in \Gamma\} \leq \sup\{[\![A_t]\!]^{\mathcal{M},\mathfrak{a}} \mid A_t \in \Delta\}$$

The standard validity and entailment are automatically defined over the terms of relational type α if 1 is replaced by 1_α and $\leq$ is a partial pointwise order induced by $\leq$ of $\{0, 1\}$. For brevity, instead of writing $\vDash A$, we will simply write A and assume its truth with respect to any standard model. In the similar way, for instance, writing $[\![A_\alpha]\!] = 1_\alpha$ assumes that α is relational and that $\vDash A$ holds.

We have defined TY logic in terms of the simply typed λ-terms and the functional interpretation based on the standard models. In addition to the standard logical operators, we have also introduced the generalized versions of Boolean connectives, which will allow us to express various lexical relations between linguistic expressions of different categories. The presented type theory could be easily interpreted on general, i.e. non-standard, frames and models in the sense of Henkin (1950). The functional interpretation of terms could also be replaced by relational semantics in the style of Muskens (1995). But since these decisions do not play an important role for our work, we have adopted the classical formulation of type theory. For instance, to obtain the intensional version of $\mathtt{TY}_1$ it is sufficient to introduce an additional basic type s for world-time pairs.[3] The fact that the current work does not model intensionality is not the reason for omitting the s type from TY logic. The reason is that the s type would unnecessarily complicate some notations and induce redundant symbols in proofs and inference rules. Nevertheless, while extending TY logic (e.g., in §2.2), we will mention how things would be in case of the intensional version of TY.

2.0.2 Semantic tableau method

A semantic tableau method, or simply a tableau method, is a proof procedure which proves theorems via refutation—a systematic attempt to find the counterexample for a theorem.[4] The tableau method is a popular proof system and many formal logics, e.g., first-order logic, modal logics and many-valued logics, have their own versions of it (D'Agostino et al., 1999). Before discussing a tableau method for TY, we would like to present a general intuition behind a tableau method and show how its propositional version works.

As we already mentioned, the tableau method is a refutation procedure, in other words, instead of showing that a formula F is valid, i.e. F is true in any model, the method tries to show that its counterexample $\neg F$ is unsatisfiable, i.e. $\neg F$ is false in any model. In the

[3] The intentional logic $\mathtt{TY}_2$ (Gallin, 1975) obtained in this way remains the same as Montague (1970)'s intesional logic but formulated in a more general and neat way.

[4] Beth (1955) and Hintikka (1955) simultaneously arrive to the idea of semantic tableau. Their versions of semantic tableau were pertinent but formulated in an unpractical manner. Beth graphically represented semantic tableau as nested two-column tables while Hintikka used a tree structure with set of formulas at nodes. Later Lis (1960) and Smullyan (1968) streamlined the method. For more historical details about the development of the tableau method see (D'Agostino et al., 1999, Ch. 1) by Fitting.

similar way, to show that a set of formulas $\mathcal{P}$ semantically entails a formula F, written as $\mathcal{P} \models F$,[5] the tableau method starts searching for a situation where the counterexample holds: all the formulas in $\mathcal{P}$ are true and F is false. Due to the search of the model for a counterexample, semantic tableau can also be regarded as a satisfiability checking procedure—given a formula, to find if there is a model that makes the formula true.

Let us now discuss how the tableau method actually searches a model for a counterexample. Throughout this work, we employ a signed tableau, which means that each formula in a tableau is paired with a truth sign, either true ($\mathbb{T}$) of false ($\mathbb{F}$), standing for the truth value of the formula. A pair of a formula and a truth sign is called a *tableau entry*. A signed tableau starts from an initial set of tableau entries. The construction process of a tableau is usually depicted as a bottom up tree growing down, where the initial entries are at the root of the tree. Based on the tree interpretation, the tableau entries are also called as *nodes*.

The construction of a tableau is carried out by applying certain schematic inference rules (i.e. *tableau rules*) to the tableau entries. A tableau rule consists of a set of *antecedent entries* and (usually one or two) consequent branches, where each branch consists of *consequent* entries (also called as *succedent* entries). The formulas in antecedent and consequent entries are called *antecedent and consequent formulas* respectively. For example, ($\mathbb{T}\wedge$) in Figure 2.1 has a single antecedent entry $\mathbb{T} : X \wedge Y$ and a single consequent branch consisting of $\mathbb{T} : X$ and $\mathbb{T} : Y$ entries. In general the antecedent formulas of a rule are longer than the consequent ones, i.e., tableau rules are inferring shorter and simpler formulas from longer and more complex ones. It is also usual that the consequent formulas of a rule are sub-formulas of the antecedent formulas; the latter dubbed as the *analyticity* property. These two common properties are necessary for efficient tableau systems. Both properties are satisfied by the rules in Figure 2.1. A rule is applicable if all its antecedent entries match to some entries on a tableau branch, and after the rule is applied, the branch is augmented with the consequent branches of the rule.

$$P \vee (Q \wedge \neg R) \models (P \vee \neg R) \wedge (\neg Q \vee Q) \tag{1}$$

The example of a tableau is presented in Figure 2.1. The tableau attempts to prove the entailment in (1). So, it starts with the counterexample represented with the nodes $\boxed{1}$ and $\boxed{2}$. The tableau is augmented with $\boxed{3}$ and $\boxed{4}$ after the rule ($\mathbb{F}\wedge$) is applied to $\boxed{2}$, where ($\mathbb{F}\wedge$) infers falsity of one of the conjuncts if the conjunction is itself false. The alternative possibilities, like falsity of one of the conjuncts, are modeled via branching. The tableau is further expanded with other rule applications following the annotations on the branches. A branch becomes closed if the closure symbol $\times$ occurs there, otherwise it is open. Closure is introduced by closure rules, e.g., by ($\times$) in case of a propositional tableau system. After a branch is closed, no rule applications are carried out on its nodes. A tableau is considered *closed* when all its branches are closed, otherwise it is *open*. The construction process continues until all tableau branches are closed or no new rule application is possible.

A branch of a tableau corresponds to the set of all possible situations that satisfies all the entries sitting on the branch, where a situation s satisfies an entry $\mathbb{X} : F$ if F is evaluated as $\mathbb{X}$ with respect to s. For instance, the most left branch in Figure 2.1 models a

[5] In other words, $\mathcal{P} \models F$ holds if and only if for every situation (i.e. model), F is true if all formulas in $\mathcal{P}$ are true.

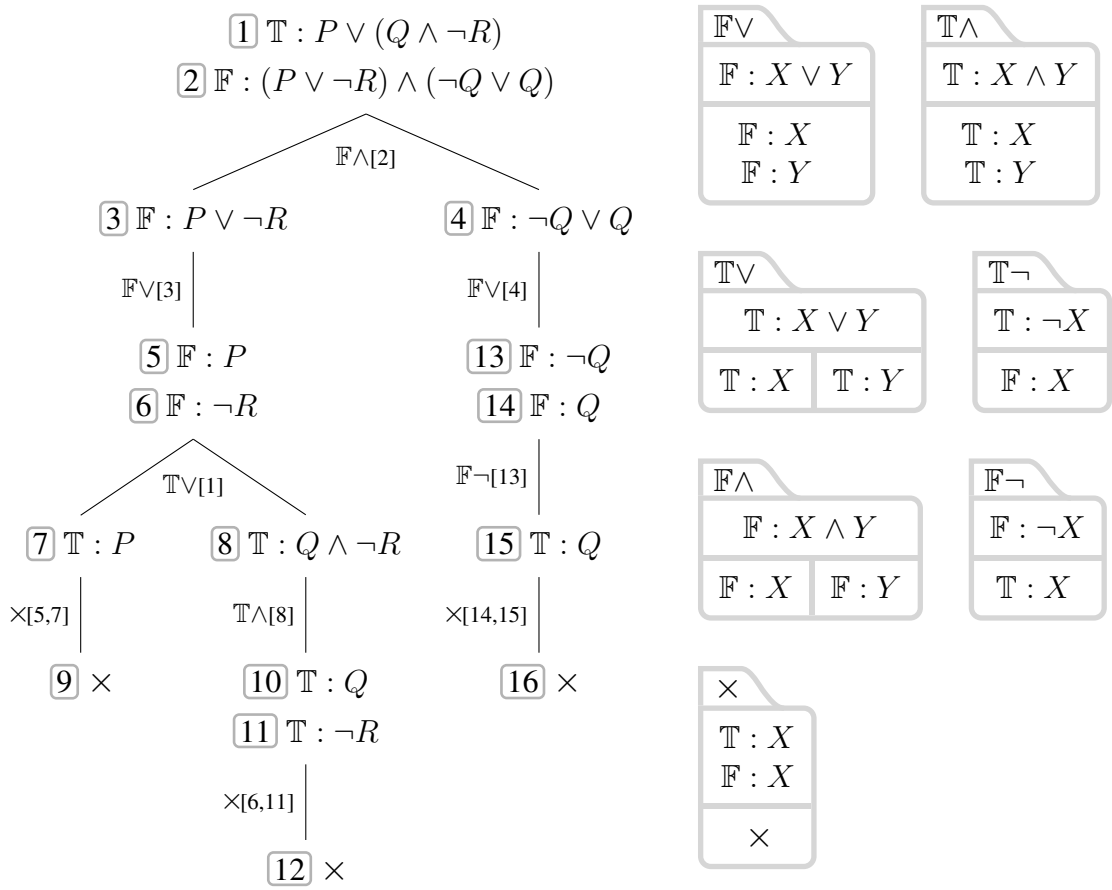


Figure 2.1: An example of a closed propositional tableau. The tableau employs the rules on the right. To clarify the tableau construction, the entries are labeled with IDs and edges are annotated with rule applications, i.e. a rule and the IDs of the entries it applies to.

set of all possible situations where $P \vee (Q \wedge \neg R)$ and P are true and $(P \vee \neg R) \wedge (\neg Q \vee Q)$, $P \vee \neg R$, P and $\neg R$ are false. Consequently, binary branching of a single branch can be seen as splitting the corresponding set of possible situations into two exhaustive subsets.[6] A closed branch does not represent any possible situation since it has the same formula with true and false signs; see ($\times$). Since all branches in a tableau share the initial entries, during the tableau construction all and only those situations are considered that satisfy the initial entries. Hence a closed tableau is understood as a failure to find a situation where the initial entries hold.

When the tableau initiated with a set of entries in (2) closes, it is said that a sequent $P_1, ..., P_m \vdash Q_1, ..., Q_n$ is correct.

$$\{\mathbb{T} : P_1, ..., \mathbb{T} : P_m,\ \mathbb{F} : Q_1, ..., \mathbb{F} : Q_n\} \tag{2}$$

A tableau system is *sound* if and only if all its rules make sound inferences. For example, the rules presented in Figure 2.1 are all sound. As a result for a sound tableau system,

[6] It is important that the subsets are exhaustive, i.e. their union represents the initial set, but they are not necessarily disjoint. For instance, when ($\mathbb{T}\vee$) is applied to an entry $\mathbb{T} : X \vee Y$ on a branch, it introduces two branches. Now if we consider the situations where both conjuncts X and Y are true, then all these situations will be still modeled by both of the resulted branches.

whenever a tableau closes, this means that the initial entries are inconsistent. In other words, $P_1, ..., P_m \vdash Q_1, ..., Q_n$ yields the following (semantic) entailment $P_1, ..., P_m \vDash Q_1, ..., Q_n$.[7] That is why the closed tableau in Figure 2.1 serves as a proof for the entailment in (1).

A tableau system is *complete* if and only if every open tableau guarantees consistency of the initial nodes. Consequently, if the entailment $P_1, ..., P_m \vDash Q_1, ..., Q_n$ is correct, then $P_1, ..., P_m \vdash Q_1, ..., Q_n$ is correct too. The sound and complete tableau system for propositional logic is represented by the set of rules in Figure 2.1.

We have introduced the main ideas behind the semantic tableau method by presenting the propositional tableau system. We also introduced the terminology and the presenting style of tableau proofs and rules. We hope that the current section provides sufficient information about the tableau method in order to follow the further discussions of tableau systems in the rest of the work. The readers who would like to know more about the tableau method are referred to D'Agostino et al. (1999) which discusses tableau systems for various formal logics.

2.1 An analytic tableau system for natural logic

The section describes Muskens (2010)'s analytic tableau system for a version of natural logic. Since the previous section provided the preliminaries for two main components of the approach, we focus here on the tableau system for natural logic. In particular, first we present employed logical forms and show what makes them natural, then we describe the format of tableau rules and demonstrate how they model various algebraic properties of lexical elements (including monotonicity). With the help of tableau proofs, we show the reasoning process over logical forms and the rules in action. But before introducing the tableau system, we briefly present the goals and intuitions behind the natural logic program and an analytic tableau method.

Natural logic is a vague notion (see § 1.3). It refers to logic that dwells in natural languages and underlies reasoning over linguistics expressions. Capturing natural logic roughly amounts to find logic that (i) is able to express linguistic semantics, (ii) characterizes all the valid inferences of natural language and (iii) formulas of which resemble linguistic surface forms. These three properties are tied up by the assumption that reasoning and the grammar of natural language are strongly related to each other in natural logic (Lakoff, 1970). For instance, Aristotelian syllogistic logic can be considered as an early version of natural logic while later developments in natural logic mainly follow van Benthem (1986); Sánchez-Valencia (1991).[8]

On the other hand, a semantic tableau method (Beth, 1955; Hintikka, 1955) is a proof system that proves a formula via attempting to refute it. If the refutation process fails then the formula is considered proved. The intuition behind a tableau method is simple: a proof

[7] The latter entailment is also referred as a *semantic sequent*. Similarly to footnote 5, we say that the entailment holds, i.e. the semantic sequent is correct, if and only if for any possible situation if all $\{P_i\}_{i=1}^m$ are true, then some Q_k from $\{Q_j\}_{j=1}^n$ is also true.

[8] For a brief history of natural logic see van Benthem (2008a). For recent contributions to natural logic see, inter alia, Fyodorov et al. (2003); Zamansky et al. (2006); van Eijck (2007); MacCartney (2009); Moss (2010b); Muskens (2010); Icard and Moss (2014).

starts with a counterexample and further expands it with the help of a predefined inventory of inference rules. While each branch in a tableau is interpreted as a set of possible situations, construction of a tableau proof can be seen as building possible situations where the initial counterexample holds.

A combination of natural logic and a tableau system was recently proposed by Muskens (2010). He adopts a fragment of higher-order logic based on the simply typed λ-calculus as a version of natural logic. By default, Muskens employs the three-sorted type theory $\mathtt{TY}_2$ of Gallin (1975) interpreted on the relational frames (Muskens, 1995, Ch. 1). But we will use here the two-sorted type theory $\mathtt{TY}_1$, i.e. without the type s for world-time pairs, with semantics over functional frames (see §2.0.1).[9] The fact that the current work does not model intensionality is not the reason for omitting s. The reason is that having the s type would unnecessarily complicate some notations and introduce redundant symbols in proofs and inference rules. Nevertheless, while extending $\mathtt{TY}_1$ logic (e.g., in §2.2), we will mention how things would be in presence of the intensional type. In regard to interpretation, our work does not hinge on the functional frames and it can be easily interpreted on relational frames, too. Notice that every function can be represented as a relation. So, for every function in a functional frame, there is a corresponding relation in a relational frame; see (Muskens, 1995, Ch. 1) for further discussion.

In order to make a version of natural logic from $\mathtt{TY}_1$, Muskens introduces constant terms in $\mathtt{TY}_1$ corresponding to lexical elements, e.g., $\mathtt{woman}_{et}$ for "*woman*". What gives a natural appearance to the $\mathtt{TY}_1$ terms is that they contain no logical connectives and quantifiers but only lexical terms, e.g., $\mathtt{and}_{ttt}$ and $\mathtt{each}_{(et)(et)t}$ for "*and*" and "*each*", respectively. The logical forms, called Lambda Logical Forms (LLFs), are the $\mathtt{TY}_1$ terms that are solely built up from variables and constant lexical terms with the help of application and λ-abstraction. Below we give several examples of LLFs along with the corresponding surface forms; the types of the lexical terms are in subscripts where p abbreviates et corresponding to the property type. Common nouns and intransitive verbs are typed as properties (et), i.e. functions from entities to propositions, and quantifiers as binary relations over properties ($(et)(et)t$). The latter typing treats quantified noun phrases as generalized quantifiers—a property over properties $(et)t$.

Some little bird flies (3)

$$\mathtt{some}_{(et)(et)t}\ (\mathtt{little}_{(et)et}\ \mathtt{bird}_{et})\ \mathtt{fly}_{et} \tag{3a}$$

Each man who dances and sings loves Mary (4)

$$\mathtt{each}_{ppt}\ \big(\mathtt{who}_{ppp}\ (\mathtt{and}_{ppp}\ \mathtt{dance}_{p}\ \mathtt{sing}_{p})\ \mathtt{man}_{p}\big)\ (\mathtt{love}_{eet}\ \mathtt{Mary}_{e}) \tag{4a}$$

Most men love some woman (5)

$$\mathtt{most}_{ppt}\ \mathtt{man}_{p}\ \big(\lambda x.\,\mathtt{some}_{ppt}\ \mathtt{womman}_{p}\ (\lambda y.\ \mathtt{love}_{eet}\ y_{e}\ x_{e})\big) \tag{5a}$$

$$\mathtt{some}_{ppt}\ \mathtt{womman}_{p}\ \big(\lambda x.\,\mathtt{most}_{ppt}\ \mathtt{man}_{p}\ (\mathtt{love}_{eet}\ x_{e})\big) \tag{5b}$$

Due to lexical terms and their compositions, which often follow a syntactic composition, most of the sub-terms of LLFs correspond to the syntactic constituents of linguistic expressions. This facilitates the automatized generation of LLFs from surface forms, as

[9] As a reminder, $\mathtt{TY}_1$ is a version of TY logic which employs only two basic types: an entity type e and a truth type t. On the other hand, $\mathtt{TY}_2$ uses an additional type s for world-time pairs.

we will see in Chapter 3. As a result, LLFs resemble linguistic surface forms and the approach can be considered as a contribution to the natural logic program. Moreover, Muskens (2011) offers to interpret LLFs as the abstract terms of Abstract Categorial Grammars (ACGs) (de Groote, 2001) or Lambda Grammars (Muskens, 2001, 2003).[10] Muskens (2010) also observes that LLFs come close to the logical forms studied in generative grammar and found in (Heim and Kratzer, 1998).

Now as the language contains a plethora of lexical terms, the goal and a challenge is to adequately model the natural semantics behind these terms. In order to reason over LLFs, Muskens employs a signed version of an analytic tableau system, in short, a *natural tableau*. A tableau entry is structured as a tuple (6) consisting of three parts: a main term, a list of terms and a truth sign.[11]

$$\underbrace{\texttt{LLF} : \texttt{argumentList}}_{\text{Binary format of a term}} : \texttt{truthSign} \tag{6}$$

While a truth sign is either true ($\mathbb{T}$) or false ($\mathbb{F}$), a main term and the elements in a list are LLFs (with an exception that the list might contain a non-lexical constant term). Examples of tableau entries are given in (3b–3e) where typing information is omitted and assumed according to (3a). The interpretation behind a tableau entry $F : \vec{A} : \mathbb{X}$ is that the term $F\vec{A}$, which is a result of applying the main term F to the elements of the argument list $\vec{A}$ (in the order of the list), is evaluated as the truth sign $\mathbb{X}$. In this way, the entries in (3b–3d) carry the same underlying semantics: "*some little bird flies*" is true is some situation, more formally $[\![\text{(3a)}]\!]^{\mathcal{M}} = 1$ for some model $\mathcal{M}$. Notice that a main term applied to the arguments in a list must result in a term of type t.

$$\texttt{some}\,(\texttt{littlebird})\,\texttt{fly} : [\,] : \mathbb{T} \tag{3b}$$
$$\texttt{some}\,(\texttt{little bird}) : [\texttt{fly}] : \mathbb{T} \tag{3c}$$
$$\texttt{some} : [\texttt{little bird},\ \texttt{fly}] : \mathbb{T} \tag{3d}$$
$$\texttt{no} : [\texttt{little bird},\ \texttt{fly}] : \mathbb{T} \tag{3e}$$

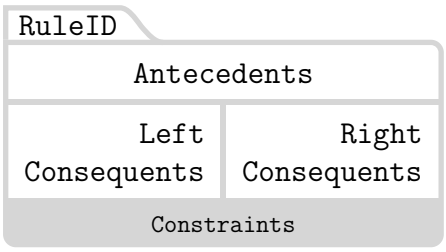
The presented entries also show how the binary format allows to represent the identical LLFs differently in a tableau. A dichotomy of a term into a main function term and its argument list is useful for two main reasons. First, it allows to traverse through a recursive structure of a term without analyzing the argument terms; see (3b–3d). Second, an argument list can be used for collecting and aligning the shared arguments of function terms. This makes it easy to contrast two LLFs by contrasting the terms in which they differ from each other. For example, with the help of this technique, the entries in (3d) and (3e) are rendered as contradictory by contrasting the quantifiers.

While presenting tableau rules, we will use a *folder format* that displays complex tableau rules in a compact way; see the sample rule (`RuleID`) below. The folder format, in addition to the standard components like antecedents and consequent branches,

[10] Though recent developments of Abstract Categorial Grammars Winter and Zwarts (2011); Groote and Winter (2015) introduce an abstract term for the closure operator while accounting event semantic. This introduction distorts the correspondence between LLFs and the abstract terms. This issue is further discussed in §2.3.1.

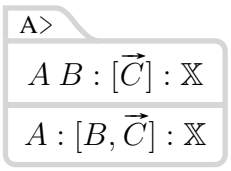
[11] Compared to the original format of tableau entries (Muskens, 2010), we reverse the order in entries. In this way, we think, the new format reflects a natural order: an argument list is on the right of a main term and in the end a truth sign evaluates the expression. This format is also better for readability when dealing with real-world examples.

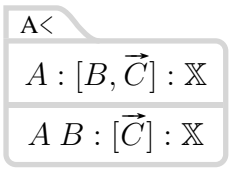
incorporates an optional bar for constraints over terms, types and truth signs. Constraints usually have a form of equations. They help to present several related rules as a single rule. By default, we assume that all typing information is carried from antecedents to consequents unless stated otherwise. For simplicity we also omit the types of common lexical terms and those that can be inferred from the well-formedness of entries. Inside a rule, the following conventions for meta-variables are adopted:

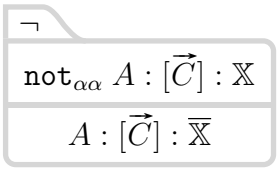


- Uppercase meta-variables match any term;
- Lowercase meta-variables match only constant terms;
- $\vec{C}$ and $\vec{c}$ match possibly an empty sequence of terms and constants respectively;
- $\mathbb{X}$ matches either $\mathbb{T}$ or $\mathbb{F}$ while $\overline{\mathbb{X}}$ stands for a negation of $\mathbb{X}$, where $\overline{\mathbb{T}} = \mathbb{F}$ and $\overline{\mathbb{F}} = \mathbb{T}$.

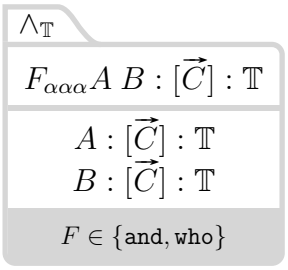
There are around 30 tableau rules presented by Muskens, which account for the Boolean connectives, determiners, the format of entries, monotonicity and related algebraic properties. For instance, the equivalence of the entries in (3b–3d) is licensed by the rules (A>) and (A<) derived from the binary format of a term.

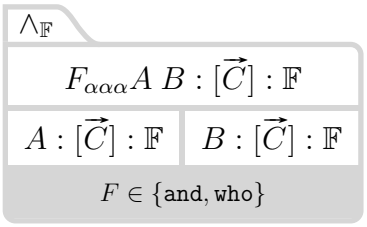


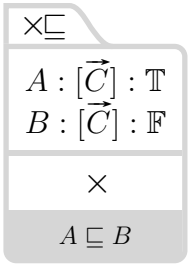




The rules for the Boolean connectives are similar to those of the propositional tableau. In natural logic these connectives have their non-Boolean counterparts which are also dealt with these rules; for example, in (¬), α can match t or et corresponding to a propositional or predicate negation respectively. Conjunctions for several types are modeled by ($\wedge_{\mathbb{T}}$) and ($\wedge_{\mathbb{F}}$), where formally speaking α can be any relational type (i.e. a type with t as its final type). For certain types there are special lexical terms for a conjunction, e.g., $\mathtt{who}_{ppp}$ is used to express a conjunction over unary predicates as in (4a).







Muskens presents ($\times\sqsubseteq$) as the only closure rule: it identifies inconsistency and after its application a tableau branch is closed. The subsumption relation $A \sqsubseteq B$, also called *inclusion*, is assumed to be provided from some Knowledge Base (KB); see §2.0.1 for the formal definition of $A \sqsubseteq B$.

The rules for quantifiers, e.g., ($\forall_{\mathbb{F}}$) and ($\exists_{\mathbb{F}}$), resemble the first-order logic tableau rules for logical quantifiers $\forall$ and $\exists$. The rules either introduce a fresh constant term on the branch or employ an old constant term from it: a term is old or fresh for a branch

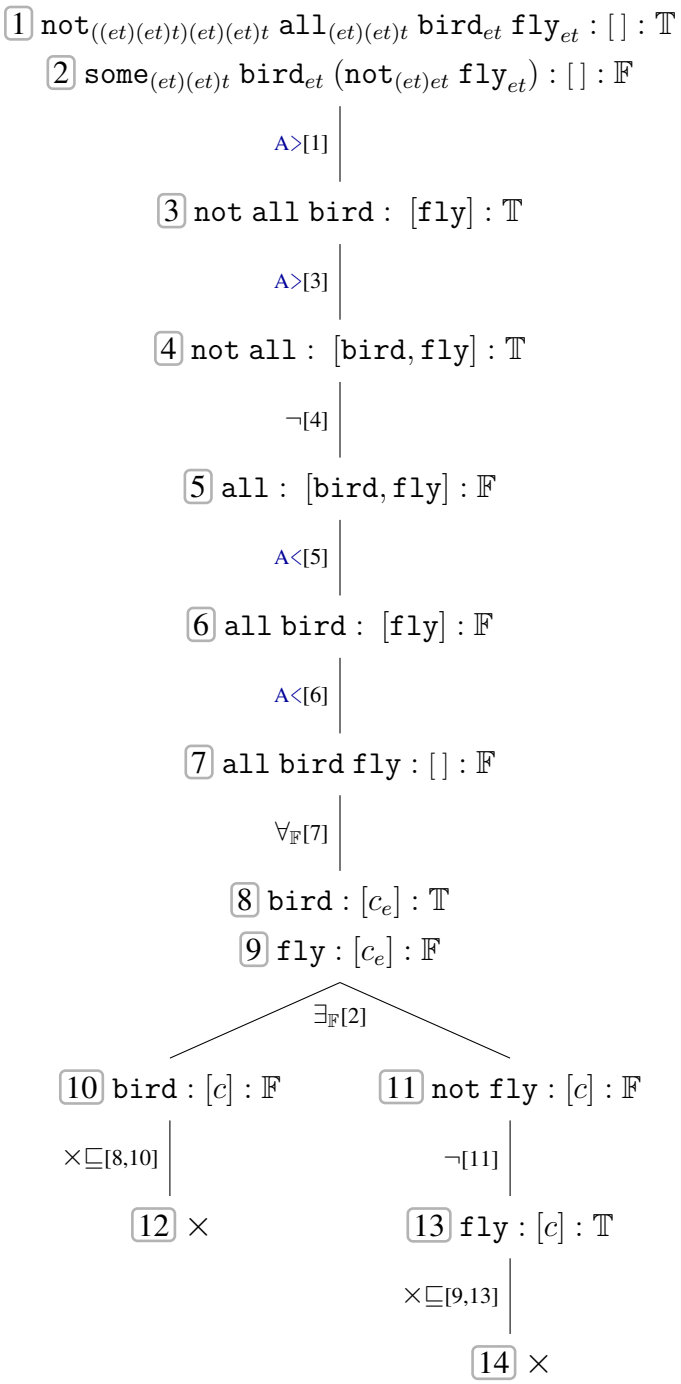
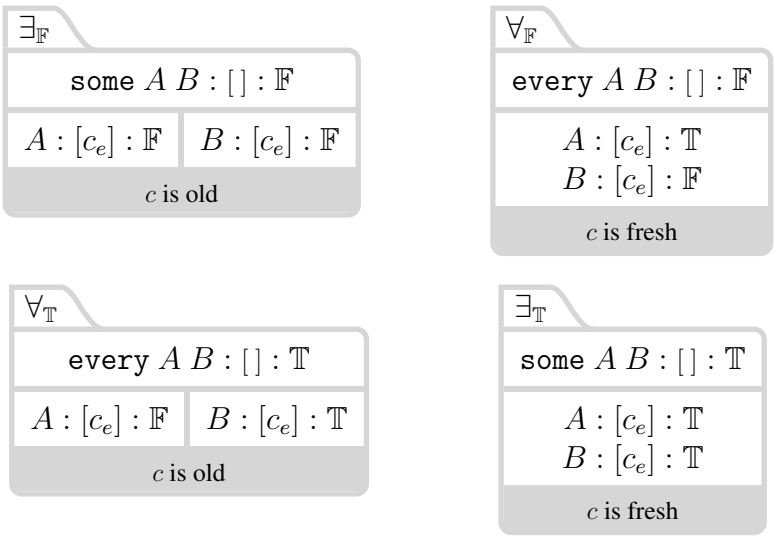


Figure 2.2: The closed tableau proves: not all bird fly $\vdash$ some bird (not fly). The types are explicated in the initial entries and they can be inferred for the rest of the entries by following the development of the tableau. To show the development process of the tableau, the entries and edges are labeled with IDs and source rule applications, respectively.

depending on whether it is already on the branch or not, respectively. The Boolean and quantifier rules in action are demonstrated by the tableau proof in Figure 2.2. For instance, the application of ($\forall_{\mathbb{F}}$) to 7 introduces a fresh constant c_e not present on the branch before. On the other hand, applying ($\exists_{\mathbb{F}}$) to 2 employs the old constant c_e already existing on the branch before the rule application.



The natural tableau system proves logical relations and checks a set of terms on consistency in a standard way. The example of a tableau proof is given in Figure 2.2. In order to prove the entailment relation, the tableau starts with a counterexample: the conclusion being false while the premise is true. In other words, the tableau starts building some model $\mathcal{M}$ where the conclusion is false and the premise true. The tableau is closed as both of its branches are rendered as inconsistent by ($\times\sqsubseteq$). The left branch is closed because the entity c is and is not `bird` at the same time. Another branch is also closed with the similar reason. The closed tableau automatically makes the sequent in (7) correct. Due to soundness of the employed rules, this implies (8). We regard the latter as a proof of "*not all birds fly*" entailing "*some bird does not fly*". Hereafter, each closed tableau will be directly understood as a proof of a certain semantic relation over natural language expressions.

$$\texttt{not all bird fly} \vdash \texttt{some bird (not fly)} \tag{7}$$

$$\texttt{not all bird fly} \models \texttt{some bird (not fly)} \tag{8}$$

It is also possible to account for monotonicity calculus in the natural tableau system (§1.3). For this it is necessary to know monotonicity properties of terms:

Definition 9 (Upward monotonicity)**.** A function term F of type $(\vec{\alpha}t)\vec{\gamma}t$ is upward monotone ($\uparrow$), denoted as $F^{\uparrow}$, if it satisfies one of the following equivalent properties:

$$\forall XY\big((X \sqsubseteq Y) \to (FX \sqsubseteq FY)\big) \tag{9}$$

$$\forall XY\big(F(X \sqcap Y) \sqsubseteq (FX \sqcap FY)\big) \tag{9a}$$

$$\forall XY\big((FX \sqcup FY) \sqsubseteq F(X \sqcup Y)\big) \tag{9b}$$

For any closed term N, the following terms are upward monotone: `every` N, `not every`, `most` N, `some`, `some` N, `and` N, `or` N, `both` N, `red` and `many` N.

Definition 10 (Downward monotonicity)**.** A function term F of type $(\vec{\alpha}t)\vec{\gamma}t$ is downward monotone ($\downarrow$), denoted as $F^{\downarrow}$, if it satisfies one of the following equivalent properties:

$$\forall XY\big((X \sqsubseteq Y) \rightarrow (FY \sqsubseteq FX)\big) \tag{10}$$

$$\forall XY\big(F(X \sqcup Y) \sqsubseteq (FX \sqcap FY)\big) \tag{10a}$$

$$\forall XY\big((FX \sqcup FY) \sqsubseteq F(X \sqcap Y)\big) \tag{10b}$$

For any closed term N, the following terms are downward monotone: `every`, `neither` N, `no`, `no` N, `few` N and `not`. The terms that are neither upward nor downward monotone are: `most`, `both`, `many`, `few` and `exactly` n, where `many` and `few` have relative semantics and n is some natural number.

Monotonicity properties are assigned to lexical items by the annotation function $\mathcal{A}$ that maps each lexical constant and an argument position to a set of some algebraic properties. For example, the monotonicity properties of `every` is encoded as $\mathcal{A}(\texttt{every}, 1) = \{\downarrow\}$ and $\mathcal{A}(\texttt{every}, 2) = \{\uparrow\}$; we write these two equations shortly as $\texttt{every} : (et)^{\downarrow}(et)^{\uparrow}t$. Based on such annotations on the types of lexical terms, it is possible to find out a monotonicity feature of a term whose main function is a lexical term as $\mathcal{A}(FA, i) = \mathcal{A}(F, i+1)$. Hence we get that `every man` is upward monotone because $\mathcal{A}(\texttt{every man}, 1) = \mathcal{A}(\texttt{every}, 2) = \{\uparrow\}$.

Taking into account the monotonicity features of terms, Muskens presents several rules where ($\uparrow\sqsubseteq$) and ($\downarrow\sqsubseteq$) are the most important ones among them. We present the rules in a slightly modified way: in the constraints we specify a concrete value of P depending on which function term is monotone. These two rules can emulate word substitution based reasoning of monotonicity reasoning. Let us have a closer look at one of them. ($\uparrow\sqsubseteq$) is a branching rule and its application yields two sets of situations. The left branch corresponds to the situations where $A \sqsubseteq B$ does not hold, i.e. $A \not\sqsubseteq B$ holds, and the right one to the situations where $G \not\sqsubseteq H$ holds. Notice that in the situations where $A \sqsubseteq B$ holds, $G \not\sqsubseteq H$ is obtained automatically due to $G^{\uparrow}$ or $H^{\uparrow}$ and the proeprty (9). Hence both branches cover all situations where the antecedent entries hold.

$\uparrow\sqsubseteq$

$G\ A : [\vec{C}] : \mathbb{T}$ $H\ B : [\vec{C}] : \mathbb{F}$	
$A : [\vec{D}] : \mathbb{T}$ $B : [\vec{D}] : \mathbb{F}$	$G : [P, \vec{C}] : \mathbb{T}$ $H : [P, \vec{C}] : \mathbb{F}$

$G^{\uparrow}$ and $P = B$, or $H^{\uparrow}$ and $P = A$

$\downarrow\sqsubseteq$

$G\ A : [\vec{C}] : \mathbb{T}$ $H\ B : [\vec{C}] : \mathbb{F}$	
$A : [\vec{D}] : \mathbb{F}$ $B : [\vec{D}] : \mathbb{T}$	$G : [P, \vec{C}] : \mathbb{T}$ $H : [P, \vec{C}] : \mathbb{F}$

$G^{\downarrow}$ and $P = B$, or $H^{\downarrow}$ and $P = A$

In order to illustrate how the rules work, we give a tableau in Figure 2.3. The tableau as usually starts with a counterexample, 1 and 2, in order to refute an entailment relation. It is further expanded using ($\uparrow\sqsubseteq$) which takes into account the upward monotonicity of one of the function terms. From the resulting branches, the left one is closed as it is identified as inconsistent with the help of the closure ($\times\sqsubseteq$) rule. The right branch is further developed by applying ($\downarrow\sqsubseteq$) to 5 and 6. The rule application takes into account the downward monotonicity of `every` or `each`. From the new branches, the right one is closed as `every` $\sqsubseteq$ `each` is assumed in the KB while the left one is grown from 8 and 9 with the rules ($\wedge_{\mathbb{T}}$) and ($\wedge_{\mathbb{F}}$) treating `who` as a conjunction. In the end, each

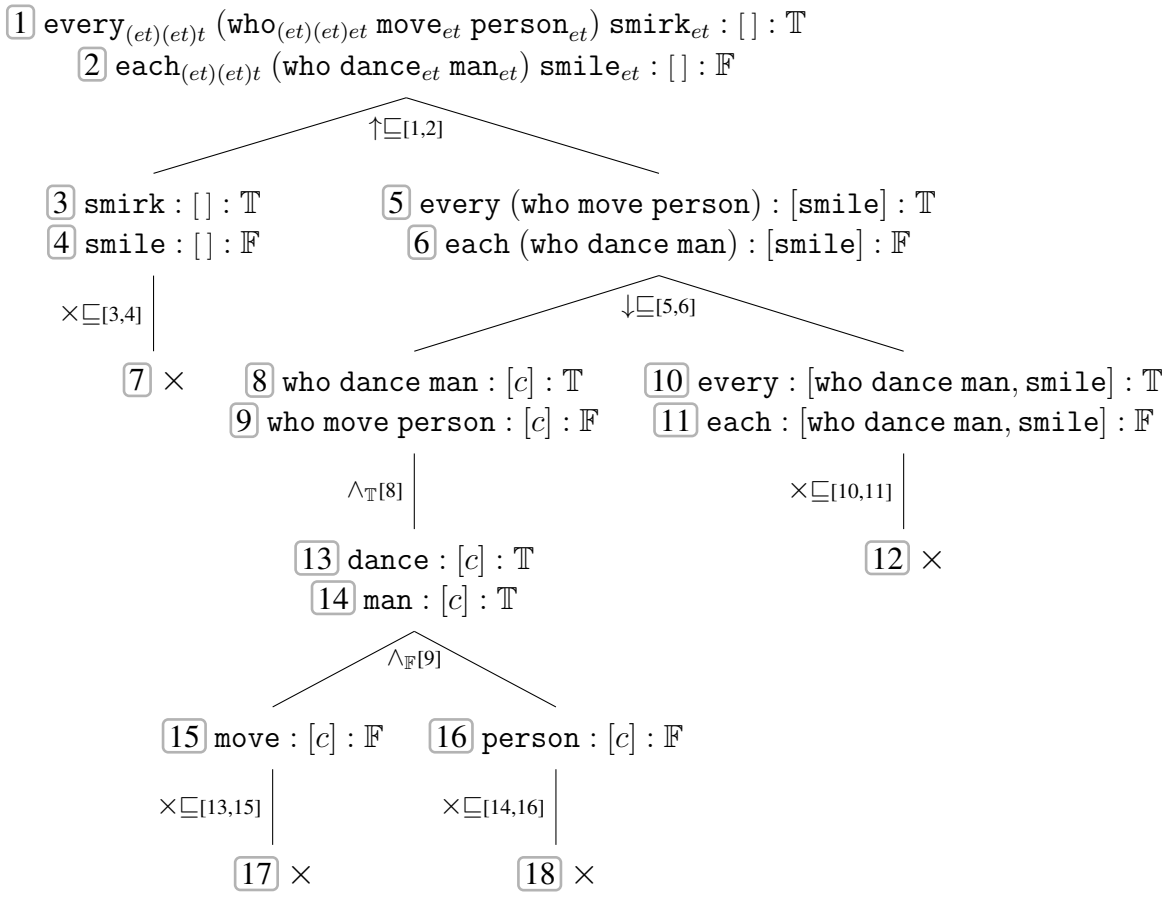


Figure 2.3: The closed tableau proves that "*every person who moves smirks*" entails "*each man who dances smiles*".

branch of the tableau is closed, i.e. the tableau is closed, which indicates a failure to find a counterexample to the entailment relation; therefore the entailment relation is proved.

The tableau proof in Figure 2.3 also shows that the natural tableau system is able to reason over phrases of the same syntactic category (not necessarily of sentential category), e.g., common nouns, noun phrases or verb phrases. For instance, the sub-trees in Figure 2.3 rooted with 5–6 and 8–9 are similar to the tableaux which prove that "*every person who moves*" entails "*each man who dances*" and "*man who dances*" entails "*person who moves*", respectively.

Muskens presents additional tableau rules that model other algebraic properties. Some of these rules are designed for the equivalent properties of monotone operators, e.g., (F$\sqcap$F$_{\mathbb{T}}$) and (F$\sqcap$F$_{\mathbb{F}}$) account for the properties in (9a) and (10b) respectively. Notice that the monotonicity constraints are placed in the antecedents as superscripts.

F$\sqcap$F$_{\mathbb{T}}$

$$\frac{F^{\uparrow}(\texttt{and}\ A\ B) : [\vec{C}] : \mathbb{T}}{\begin{array}{c} FA : [\vec{C}] : \mathbb{T} \\ FB : [\vec{C}] : \mathbb{T} \end{array}}$$

F$\sqcap$F$_{\mathbb{F}}$

$$\frac{F^{\downarrow}(\texttt{and}\ A\ B) : [\vec{C}] : \mathbb{F}}{\begin{array}{c} FA : [\vec{C}] : \mathbb{F} \\ FB : [\vec{C}] : \mathbb{F} \end{array}}$$

The rules (F⊓⊔$\mathrm{F}_{\mathbb{F}}$) and (F⊔$\mathrm{F}_{\mathbb{T}}$) model a function F that is splitting (9b′) and have meet (9a′) respectively. For any closed term N, `some` and `some` N are splitting while `every` N and `both` N have meet. The properties (9a′) and (9b′) complement the implications in (9a) and (9b) up to equivalence.

$$\forall XY\big((FX \sqcap FY) \sqsubseteq F(X \sqcap Y)\big) \qquad \textit{having meet} \qquad (9a')$$

$$\forall XY\big(F(X \sqcup Y) \sqsubseteq (FX \sqcup FY)\big) \qquad \textit{being splitting} \qquad (9b')$$

F⊓⊔$\mathrm{F}_{\mathbb{F}}$

$$\frac{F(\texttt{and}\ A\ B) : [\vec{C}] : \mathbb{F}}{FA : [\vec{C}] : \mathbb{F} \qquad FB : [\vec{C}] : \mathbb{F}}$$

F has meet

F⊔$\mathrm{F}_{\mathbb{T}}$

$$\frac{F(\texttt{or}\ A\ B) : [\vec{C}] : \mathbb{T}}{FA : [\vec{C}] : \mathbb{T} \qquad FB : [\vec{C}] : \mathbb{T}}$$

F is splitting

The latter four rules and the analogous rules for the similar algebraic properties can be seen as the rules discarding the Booleans operators: while antecedents contain Booleans, the consequents are free from them. (⊓F) and (λ⊓) are also introduced by Muskens to give a wide scope to Booleans and make their removal easier. The rules are of distributive characteristics. In (⊓F), the argument distributes over conjunct functions while in (λ⊓) λ-abstraction distributes over embedded conjuncts. Corresponding rules are available for `or` and `not` too. For checking the soundness of these rules, the definition of $A \sqcap B$ suffices.

⊓F

$$\frac{(\texttt{and}\ F\ G)\ A : [\vec{C}] : \mathbb{X}}{\texttt{and}\ (FA)(GA) : [\vec{C}] : \mathbb{X}}$$

λ⊓

$$\frac{(\lambda x.\ \texttt{and}\ A\ B) : [\vec{C}] : \mathbb{X}}{\texttt{and}\ (\lambda x.\ A)\ (\lambda x.\ B) : [\vec{C}] : \mathbb{X}}$$

At this point we finish presenting the tableau rules of Muskens (2010). We have discussed most of the rules. For the complete list of them we refer readers to the original work. In the next sections we introduce additional rules having a formal logical nature, similarly to the currently presented ones. In Chapter 4, we will introduce a bunch of rules that are more of *linguistic* nature and which are necessary for wide-coverage reasoning.

We have presented the main ides behind the natural tableau of Muskens (2010) according to which (i) LLFs play proxies for semantic representations and surface forms at the same time and (ii) there is a tableau proof system that operates on LLFs. We have showed how reasoning is carried out in the tree style and discussed a plethora of tableau rules mainly concerning the Boolean operators and monotone functions.

Informally speaking, Muskens regards natural language as a formal logical language and models natural reasoning in the same style as it is done for formal logics in terms of proof systems. At first glance this approach is in the same vein as the proposal by Montague (1970) that there is *no important theoretical difference between natural languages and the artificial languages of logicians*, but they differ in terms of focus. While Montague concentrates on the complete translation of linguistic expressions into some well-studied formal logic, Muskens focuses more on the development of the proof calculus over natural-looking LLFs rather than on translation.

The advantages of the natural tableau system are several. The underlining logic, due to its higher-order nature, models certain linguistic phenomena in a more natural way than first-order logic offers, e.g., it can easily account for generalized quantifiers (Montague, 1973; Barwise and Cooper, 1981), monotonicity calculus (van Benthem, 1986; Sánchez-Valencia, 1991; Benthem, 1991; Icard and Moss, 2014) and subsective adjectives. Since LLFs resemble syntactic trees of linguistic expressions to some extent, from the application point of view, obtaining LLFs from raw text is cheaper than obtaining first-order logic formulas. In particular, Chapter 3 will show that producing LLFs is almost parsing. Moreover, the natural tableau handles reasoning over any finite number of entries, which is beyond any alignment-based approach to reasoning and monotonicity calculus as such.

In the next sections we propose an extension of the natural tableau in several dimensions: extending the type system of $\mathtt{TY}_1$, expanding the format of tableau entries and increasing the inventory of tableau rules. These changes aim to make the natural tableau more robust and suitable for reasoning over LLFs of wide-coverage text.

2.2 Extending the type system

We shall argue that the syntactic categories of lexical terms are valuable information for guiding tableau construction. Unfortunately, the types built upon the e and t basic types are not informative from syntactic point of view. As a solution, we offer a conservative extension of the TY type system by incorporating extra types motivated by natural language syntax.

The presented natural tableau uses LLFs that are simply typed λ-terms over the type system $\mathscr{T}_{\mathtt{Sem}}$ where $\mathtt{Sem} = \{e, t\}$. We shall refer the types of $\mathscr{T}_{\mathtt{Sem}}$ as *semantic types* because they are semantically motivated and are usually used for typing the terms representing semantics. We argue that the semantic types are poor in guiding the rule applications in the natural tableau system. The reason is that the semantic types treat several different syntactic categories in a uniform way. For instance, nouns, intransitive verbs and prepositional phrases (PPs) are usually treated as unary predicates of type et; transitive verbs and prepositions as binary predicates of type eet; and adverbs, adjectives and relative nouns as binary predicates of type $(et)et$.

Let us consider the node in (11) where h_{et} is some lexical term. Based only on its type, we can infer that h_{et} is either a noun or a verb. The similar situation is with the LLF in (12). It can represent at least two different syntactic constructions: an adjective-noun pair (12a) or an adverb-verb pair (12b). But if we plan to unfold semantics of verbs differently from nouns, e.g., to introduce an event entity for a verb, then based on semantic types it is not clear how to decompose the terms. Obviously we would also like to analyze the entries (13a) and (13b) in different ways as their LLFs model the semantics of completely different constructions. But distinguishing the entries requires at least some sort of mechanism for identifying prepositions.

$$h_{et} : [c_e] : \mathbb{T} \tag{11}$$

$$A_{(et)et}\ B_{et} : [c_e] : \mathbb{T} \tag{12}$$

$$\mathtt{little}_{(et)et}\ \mathtt{bird}_{et} : [c_e] : \mathbb{T} \tag{12a}$$

$$\mathtt{high}_{(et)et}\ \mathtt{fly}_{et} : [c_e] : \mathbb{T} \tag{12b}$$

$$a_{(et)et}(b_{eet}c_e) : [c_e] : \mathbb{T} \tag{13}$$

$$\texttt{quietly}_{(et)et}\,(\texttt{follow}_{eet}\ \texttt{john}_e) : [c_e] : \mathbb{T} \tag{13a}$$

$$\texttt{wife}_{(et)et}\,(\texttt{of}_{eet}\ \texttt{john}_e) : [c_e] : \mathbb{T} \tag{13b}$$

Syntactic information is also relevant for the project of natural logic. In particular, there is a tight connection, or sometimes a coincidence, between the syntactic rules and the rules relating surface forms to logical forms (Lakoff, 1970). Taking into account that unfolding semantics of LLFs is a duty of tableau rules, they can also be rendered as the rules that relate LLFs to some deeper semantic analyses. Hence, there is a good reason to believe in the tight connection between tableau rules and syntactic rules. The latter makes syntactic information crucial for tableau rules.

Yet another argument for the importance of syntactic information is motivated by natural reasoning and its efficiency. For humans it is easy to use lexical relations like hypernymy, synonymy and antonymy while reasoning on natural language text. Notice that the lexical items in these relations are usually of the same syntactic category. For instance, it is smarter to contrast two lexical terms that model lexical elements of the same syntactic category than two lexical terms of the same semantic type. Thus, we believe that the information about syntactic categories can guide to short proofs by giving the priority to those rule applications that contrast or align the terms of the same syntactic category.

While being in need of syntactic information, one can think of a scenario where each lexical term comes with a Part of Speech (POS) tag. Then, it is possible to design some procedure $SyntCat$ that identifies the syntactic category of a lexical term $h_\alpha^{\texttt{POS}}$ based on its POS tag and type. Unfortunately, it is not easy to extend this procedure over compound terms. For instance, in order to find out whether a term $a_{(et)et}b_{et}$ is an adjective-noun (12a) or adverb-verb (12b) pair, $SyntCat$ needs to be able to calculate the syntactic category from the POS tags and the types of the constituent lexical terms. Taking into account the number of combinations of types, POS tags, and λ-term forming operations, the procedure $SyntCat$ becomes complicated. In the natural tableau, such an ad hoc and complex procedure for determining the syntactic category of an LLF is unwanted for two main reasons. First, the tableau rules heavily rely on LLF matching when properly identifying their antecedent entries. Using $SyntCat$ in the constraints of the rules will make the natural tableau theory *unnecessary cumbersome* for presentation. Second, since one of our goals is to implement the theorem prover for the natural tableau, we anticipate that integration of $SyntCat$ in the LLF matching procedure will cause *significant inefficiency* in theorem proving.

Fine-grained efficient matching over LLFs is crucial for the natural tableau system. Also as we have seen from the examples (11–13) that the semantic types are not enough informative to smoothly guide the rule application in the wide-coverage version of natural tableau. As a solution we suggest to extend the type system $\mathscr{T}_{\texttt{Sem}}$ by adding new basic types to the semantic ones. Particularly, we introduce the basic syntactic types `np` (for noun phrases), `n` (for nouns), `s` (for sentences), `pp` (for prepositional phrases) and `pr` (for particles). The syntactic types are motivated by the basic syntactic categories NP, N, S and PP of Combinatory Categorial Grammar (CCG) (Steedman, 2000).[12] Hence, $\mathscr{T}_{\texttt{SS}} \stackrel{\text{def}}{=} \mathscr{T}_{\texttt{Sem}\cup\texttt{Syn}}$ is a new type system where $\texttt{Syn} = \{\texttt{np}, \texttt{s}, \texttt{n}, \texttt{pp}, \texttt{pr}\}$. The extension of

[12]The usage of CCG categories by the CCG parsers (Clark and Curran, 2007; Lewis and Steedman,

the type system automatically yields the extended type logic, referred as $\mathtt{TY}_{\mathtt{SS}}$. A type is called *semantic* or *syntactic* if it belongs to $\mathscr{T}_{\mathtt{Sem}}$ or $\mathscr{T}_{\mathtt{Syn}}$, respectively. A term is called a *syntactic term* if it is built up only from syntactic terms, where constant and variable terms of syntactic type are syntactic terms. A *semantic term* is also defined similarly. Hereafter, we will use a boldface style for lexical syntactic terms. The syntactic LLF in (15) is a syntactic counterpart of the semantic LLF in (14), where the vp type abbreviates $(\mathtt{np}, \mathtt{s})$.

$$\mathtt{no}_{(est)(est)st}\ \mathtt{bird}_{est}\ \mathtt{fly}_{est} \tag{14}$$

$$\mathbf{no}_{\mathtt{np},\mathtt{vp},\mathtt{s}}\ \mathbf{bird}_{\mathtt{n}}\ \mathbf{fly}_{\mathtt{vp}} \tag{15}$$

Interaction between the syntactic and semantic types is established by a *subtyping relation* ($<:$). We say that α is a subtype of β, or β consumes α, and write $\alpha <: \beta$.

Definition 11 (Subtyping)**.** The subtyping relation $<:$ is the smallest partial order over $\mathscr{T}_{\mathtt{SS}}$ satisfying the following properties:[13]

(a) $e <: \mathtt{np}$, $\quad \mathtt{s} <: t$, $\quad \mathtt{n} <: et$, $\quad \mathtt{pp} <: et$;

(b) For any $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathscr{T}_{\mathtt{SS}}$, $\quad (\alpha_1, \alpha_2) <: (\beta_1, \beta_2)$ $\quad$ iff $\quad \beta_1 <: \alpha_1$ and $\alpha_2 <: \beta_2$

According to the definition of subtyping, for example, $\mathtt{vp} <: et$, i.e. the type of intransitive verbs is subsumed by the type of unary predicates, and $(\mathtt{np}, \mathtt{vp}) <: (eet)$, i.e. the type of transitive verbs is a subtype of the type of binary predicates. In connection with the subtyping relation, we also introduce an additional subtyping clause: (iv) for $\mathtt{TY}_{\mathtt{SS}}$ frames in Definition 4 and (vii) for $\mathtt{TY}_{\mathtt{SS}}$ terms in Definition 2 (see §2.0.1):

(iv) if $\alpha <: \beta$, then $\mathcal{D}_\alpha \subseteq \mathcal{D}_\beta$ *Subtyping for frames*

(vii) if $\alpha <: \beta$, then $\mathcal{T}_\alpha \subseteq \mathcal{T}_\beta$ (i.e. each term A_α is of type β). *Subtyping for terms*

From the new subtyping clauses it follows that the term $A_\alpha B_\beta$ is of type γ if $\alpha <: (\beta, \gamma)$. Also both LLFs in (14) and (15) are of type t as the LLF in (15) is of type $\mathtt{s}$, hence of type t. Now a term can be of several types, where all these types for a complete lattice ordered with $(<:)$. For example, a term $\mathbf{love}_{\mathtt{np},\mathtt{vp}}$, apart from an eet type, is of four other types where $(\mathtt{np}, \mathtt{np}, \mathtt{s})$ and eet are the least and the greatest types, respectively. We refer this property of typing as *multiple typing*.

Now, with the help of the multiple typing, it is straightforward to apply $\mathbf{love}_{\mathtt{np},\mathtt{np},\mathtt{s}}\ \mathbf{mary}_{\mathtt{np}}$ directly to c_e, and there is no need for introducing new terms $\mathtt{love}_{eet}$ and $\mathtt{mary}_e$. For the same reason it is not necessary to introduce $\mathtt{man}_{et}$ for applying it to c_e as $\mathbf{man}_{\mathtt{n}} c_e$ is already a well-typed term. The latter examples show that syntactic terms not only carry the syntactic information but also some of them emulate their semantic counterparts. For more evidence consider the entries in (12a′–13b′) that are syntactic counterparts of the entries in (12a–13b).

$$\mathbf{little}_{\mathtt{n},\mathtt{n}}\ \mathbf{bird}_{\mathtt{n}} : [c_e] : \mathbb{T} \tag{12a′}$$

2014a) and CCGbank (Hockenmaier and Steedman, 2007) will also facilitate the process of obtaining LLFs with syntactic types. Notice that pr type corresponds to the CCG category *PR* for particles, which was introduced later in rebanked CCGbank (Constable and Curran, 2009; Honnibal et al., 2010). We include the pr type in the type system in order to make the natural tableau theory and prover compatible with the CCG derivations allowing the *PR* category.

[13] In case of the intensional type system $\mathtt{TY}_2$, with the s type, the first clause would be altered according to $\mathtt{s} <: st$, $\mathtt{n} <: est$ and $\mathtt{pp} <: est$.

$$\mathbf{high}_{\mathrm{vp,vp}}\ \mathbf{fly}_{\mathrm{vp}} : [c_e] : \mathbb{T} \qquad (12b')$$

$$\mathbf{quietly}_{\mathrm{vp,vp}}\ (\mathbf{follow}_{\mathrm{np,vp}}\ \mathbf{john}_{\mathrm{np}}) : [c_e] : \mathbb{T} \qquad (13a')$$

$$\mathbf{wife}_{\mathrm{n,n}}\ (\mathbf{of}_{\mathrm{np,pp}}\ \mathbf{john}_{\mathrm{np}}) : [c_e] : \mathbb{T} \qquad (13b')$$

Nevertheless, sometimes it is inevitable to introduce a semantic term since its syntactic counterpart is not able to give adequate semantics. For instance, if we have the entry $\mathbf{red}_{\mathrm{n,n}}\mathbf{car}_{\mathrm{n}} : [c_e] : \mathbb{T}$ in a tableau branch, then one has to introduce the semantic term $\mathtt{red}_{et}$ in order to assert the redness of c_e by the entry $\mathtt{red}_{et} : [c_e] : \mathbb{T}$, because the term $\mathbf{red}_{\mathrm{n,n}}c_e$ is not well-formed. Notice that the introduction of a new term $\mathtt{red}_{et}$ would be inevitable in case of using only semantic types too as $\mathtt{red}_{(et)et}$ cannot serve as a unary predicate.

Incorporating terms of syntactic and semantic types in one system can be seen as putting two different representation levels together. These levels are very similar to the abstract and semantic levels found in ACG (de Groote, 2001) or Lambda Grammar (Muskens, 2003): the syntactic and semantic terms can be seen as the terms of the abstract and semantic levels, respectively, while the subtyping relation as the reminiscence of the type homomorphism from the abstract types to the semantic types. Along with the similarities, there are also differences. Recent developments of ACG (Winter and Zwarts, 2011; Groote and Winter, 2015) employ the semantic type v for events and do not include the abstract types for pp and pr. Also the terms consisting of both abstract and semantic terms are not allowed in the ACG and Lambda Grammar formalisms.

We have shown that the semantic types cannot provide LLFs with the syntactic information, which is so important for fine-grained LLF matching, the project of natural logic and efficient natural reasoning. As a solution we introduced the syntactic types, borrowed from CCG formalism, in the type system. The syntactic types automatically provide the terms with the syntactic information, and at the same time, many semantic terms can be emulated by the syntactic terms. As a result we augment LLFs with the syntactic categories without losing their natural appearance. Moreover, the tableau rules and proofs presented in the previous section are accommodated in the extended type theory in a straightforward way. For the demonstration, compare the tableau proof for the $\mathtt{TY}_1$ terms (Figure 2.2) to the proof for the corresponding $\mathtt{TY}_{\mathtt{SS}}$ terms (Figure 2.7 in Appendix A).

The proposed solution gives robustness to the natural tableau. Now it is simpler to obtain LLFs with syntactic types from the CCG derivations produced by CCG parsers. Moreover, LLFs with syntactic types offer much more facilities for natural language generation compared to information-poor semantic types. For example, if the natural tableau system generates conclusions from premises or a counterexample for textual entailments, it will be easier to generate linguistic expressions from LLFs of syntactic types.

2.3 Extending the tableau entries

An adjunct-head construction is one of the most common syntactic construction in natural languages. It is crucial to account for this phenomenon in the natural tableau system. In this section, we argue for adding an extra slot to the format of tableau entries. The new slot will serve as a storage for modifiers of nouns and verbs. The storage will make it easy to link a remote modifier to its head by saving it temporarily. This technique also contributes to accommodate event semantics in LLFs. We start with discussing adverbial

phrases that modify verb phrases (VPs). In particular, we show how event semantics can be integrated in LLFs and the tableau system. Then we demonstrate how modifiers-noun pairs can be accommodated in the proposed approach. As we will see in the end, the offered solution contributes to natural appearance of LLFs, the reasoning power of the tableau system and efficiency of theorem proving.

2.3.1 Event semantics and LLFs

We start with a discussion on the problems of accommodating event semantics in LLFs. First, two approaches for pairing the standard compositional semantics and event semantics, by Winter and Zwarts (2011) and Champollion (2010), are discussed. Then we argue that the representations from these approaches are not appropriate for being LLFs. The reason is their unnatural appearances and unnecessarily complicated type systems.

Main properties of modifiers are their *optionality* and the ability to *iterate*. The iteration of adverbial modifiers is demonstrated in (16). Semantics of the sentences with adverbial modifiers are elegantly modeled in first-order logic (16a) using Davidsonian events (Davidson, 1967). Each verb is represented by an event entity that is related to its participants via the corresponding verbal predicate. Adverbial modifiers then act as unary predicates and modify the event entity. By modeling adverbials as conjuncts, Davidsonian semantics easily captures the optionality and iteration properties of adverbial modifiers and the entailments licensed by these properties. A refined version of Davidsonian semantics (Parsons, 1990) is also presented in (16b). This type of analysis, often called neo-Davidsonian, directly accounts for optional verbal arguments: a verbal argument modifies the event via the corresponding thematic relation, e.g., AGENT, which itself is just another property of the event. In the end, there is no need for several verbal predicates differing in terms of the argument structures in order to model a verb with optional arguments, e.g., "*eat*" in "*John ate*" and "*John ate an apple*".

John jogged slowly in Tilburg at midnight (16)

$\exists e\big(\texttt{jog}(\texttt{John}, e) \wedge \texttt{slow}(e) \wedge \texttt{in\_Tilburg}(e) \wedge \texttt{at\_midnight}(e)\big)$ (16a)

$\exists e\big(\texttt{jog}(e) \wedge \texttt{AGENT}(\texttt{John}, e) \wedge \texttt{slow}(e) \wedge \texttt{in\_Tilburg}(e) \wedge \texttt{at\_midnight}(e)\big)$ (16b)

$\exists\text{-CLOS}_{V,S}\left(\text{AT\_MIDNIGHT}_{V,V}\left(\text{IN\_TILBURG}_{V,V}\left(\text{SLOWLY}_{V,V}\left(\text{JOG}_{NP,V}\ \text{JOHN}_{NP}\right)\right)\right)\right)$ (16c)

$[\![\text{CLOSURE}]\!]\left([\![\text{AG}]\!]\ [\![\text{john}]\!]\left([\![\text{slowly}]\!]\left([\![\text{in\_Tilburg}]\!]\left([\![\text{at\_midnight}]\!]\ [\![\text{jog}]\!]\right)\right)\right)\right)$ (16d)

$\textbf{at\_midnight}_{\text{vp,vp}}\big(\textbf{in\_Tilburg}_{\text{vp,vp}}\big(\textbf{slowly}_{\text{vp,vp}}\textbf{jog}_{\text{vp}}\big)\big)\textbf{John}_{\text{np}}$ (16e)

Pairing (neo-) Davidsonian event semantics and compositional semantics poses two challenges: the event modification and the event quantification problems (Winter and Zwarts, 2011). *The event modification problem* is about making sure that an adverbial modifies a correct event entity and no other entities. *The event quantification problem* concerns a correct scope of an event quantifier, for instance, to get correctly the narrow scope of an event quantifier, as in (17a) for the sentence in (17). Notice that the wide scope of the event quantifier in (17a) would give wrong semantics for (17).

Nobody kissed Mary passionately (17)

$\neg\exists x\big(\exists e\big(\texttt{kiss}(\texttt{Mary}, x, e) \wedge \texttt{passionately}(e)\big)\big)$ (17a)

$\neg\exists x\big(\exists e\big(\texttt{kiss}(e) \wedge \texttt{AGENT}(x, e) \wedge \texttt{THEME}(\texttt{Mary}, e) \wedge \texttt{passionately}(e)\big)\big)$ (17b)

$$\text{NOBODY}_{(NP,S),S}\left(\lambda x.\ \exists\text{-CLOS}_{V,S}(\text{PASSIONATELY}_{V,V}(\text{KISS}_{NP,NP,V}\,\text{MARY}_{NP}\,x_{NP}))\right) \quad (17c)$$

$$[\![\text{CLOSURE}]\!]\ \left([\![\text{AG}]\!]\ [\![\text{nobody}]\!]\ \left([\![\text{TH}]\!]\ [\![\text{Mary}]\!]\ \left([\![\text{passionately}]\!]\ [\![\text{kiss}]\!]\right)\right)\right) \quad (17d)$$

$$\textbf{nobody}_{\text{vp,s}}\ \left(\textbf{passionately}_{\text{vp,vp}}\ \left(\textbf{kiss}_{\text{np,vp}}\ \textbf{Mary}_{\text{np}}\right)\right) \quad (17e)$$

Since we plan to model adverbials, we would like to incorporate event semantics and its expressive power into the tableau system. But what the LLFs, e.g., of (16) and (17), should look like then? One of the key features of LLFs is to be similar to linguistic forms and therefore *easily obtainable* from them. For example, it is obvious that first-order logic formulas are not suitable candidates for LLFs: (16a) and (16b) are far from the linguistic form in (16). Since we have emphasized the tight connection between LLFs and the abstract terms of ACG, it is worthy to consider them as possible candidates for LLFs.

An abstract term of ACG that can yield (16a) or (16b) on the semantic level is given in (16c).[14] Notice that a saturated verb is of abstract type V and its semantic interpretation is a set of events. The terms of type V are converted into sentential terms with the help of the *existential closure operator* ($\exists$-CLOS) of type (V, S). The semantic counterpart of the operator converts a set of events into a truth value, respectively. On the linguistic level (i.e. pheno-level), the operator is treated as vacuous. The existential closure operator does not always take the widest scope as it is in (16c). In opposite, it takes the narrowest scope compared to other quantifiers, e.g., see the abstract term in (17c). Unfortunately, we are not aware of any work that identifies the existential closure operator with any lexical element that fits into the compositional architecture of ACG. For this reason, we find the terms similar to $\exists$-CLOS as a hinder for a natural appearance of LLFs. Moreover, from the natural logic point of view, the distinction between the types of saturated verbs and the sentences can be seen as an unnecessary complication of logical forms.

Another noteworthy combination of standard compositional semantics and event semantics is due to Champollion (2010, 2014). According to his approach, the modifiers and arguments of a VP are both treated as VP modifiers, where the arguments are augmented with thematic relations.[15] The semantics of (16) and (17), represented in first-order logic as in (16b) and (17b), are obtained by combining the semantic recipes of the lexical entries in (16d) and (17d), respectively. We refer the latter structures as *compositional* terms since they express the instructions for semantic composition. The continuation of an event is *closed* by the *closure operator* [[CLOSURE]] of type vt and the term of truth type is returned in the end. Unlike $\exists$-CLOS of Winter and Zwarts (2011), the operator [[CLOSURE]] always applies last; for the contrast compare (17c) to (17d).

We do not consider the compositional terms of Champollion (2014) as adequate proxies for LLFs at least for two reasons. First, the terms contain *abstract* lexical items, e.g., AG and TH, corresponding to thematic relations. These abstract items make the terms *un-*

[14] Winter and Zwarts (2011) and Groote and Winter (2015) model event semantics in ACG using the additional abstract type V that translates into the semantic level as the type (vt), where v is the event type. After the introduction of the type V, abstract terms for verbs and generalized quantifiers now differ in their final type. The verbs return V (e.g., $\text{JOG}_{NP,V}$) and the generalized quantifiers S (e.g., $\text{NOBODY}_{(NP,S),S}$).

[15] Every subcategorization of VP is semantically interpreted as a generalized quantifier over events, i.e. of semantic type $((vt)t)$, where v is the type for events. In other words, they denote continuations of events. VP modifiers are of type $((vt)t)(vt)t$ consequently. The narrowest scope of an event quantifier is achieved by interpreting VP projections as a continuation of events and by including the existentially quantified event in the denotation of a lexical verb.

natural and difficult to be automatically obtained from the English text.[16] Second, though the approach of Champollion (2014) is quite powerful, we find the types in compositional terms too complex for natural reasoning. It is not obvious that reasoning over compositional terms is more robust or powerful than reasoning over the first-order logic formulas, e.g., (16b) and (17b), which are directly obtained from them by substituting the entries with the semantic denotations.[17]

We have examined two semantic representations, originating from Winter and Zwarts (2011) and Champollion (2010), as the candidates for LLFs. Our main objections are related to the abstract non-lexical elements they employ and the type systems that are *unnatural* for natural reasoning. In the next subsection, we propose our alternative solution to incorporate event semantics into LLFs.

2.3.2 Modifier list and event semantics

The main difficulty in encoding event semantics in LLFs is that events are abstract entities that are usually invisible on the surface level. On the other hand, we want LLFs to resemble surface forms in order to capture some logical relations in a swift way and to be easily obtainable for surface forms. Here, we will show that this is possible. In particular, we induce event semantics from LLFs in a procedural way while LLFs still having a natural appearance, e.g., (16e) and (17e).

John jogged slowly in Tilburg at midnight	(16)
$\mathbf{at\_midnight}_{\mathrm{vp,vp}}\big(\mathbf{in\_Tilburg}_{\mathrm{vp,vp}}\big(\mathbf{slowly}_{\mathrm{vp,vp}}\mathbf{jog}_{\mathrm{vp}}\big)\big)\mathbf{John}_{\mathrm{np}}$	(16e)
Nobody kissed Mary passionately	(17)
$\mathbf{nobody}_{\mathrm{vp,s}}\ \big(\mathbf{passionately}_{\mathrm{vp,vp}}\ \big(\mathbf{kiss}_{\mathrm{np,vp}}\ \mathbf{Mary}_{\mathrm{np}}\big)\big)$	(17e)

We can achieve pairing LLFs and event semantics with the help of the tableau system. There are basically no restrictions on tableau rules except that the consequent entries of a rule have to be semantically entailed from the antecedent entries. So, it is possible that tableau rules introduce event entities, like fresh entities, during tableau construction. In order to give the intuition what these rules might look like, let us first consider the rule ($\mathrm{EV}'_{\mathbb{T}}$), where α^k stands for a vector type that consists of the k number of αs. The rule has a dark gray background as it represents a temporary rule that serves as a demonstration.

[16] Maybe for the natural languages that have a rich case system, the lexical items for thematic relations can be successfully identified as cases markers. But for English, which almost lost its case system, these lexical items indeed come out of nowhere.

[17] Additionally, the monotonicity properties of generalized quantifiers (GQs) are not accessible in compositional terms of Champollion (2014) as noun phrases (NPs) are arguments of thematic relations. In order to enable the monotonicity properties of GQs, one can type-raise NP arguments from $(et)t$ to $\big(((et)t)cc\big)cc$, where c abbreviates $(vt)t$. In this way, $[\![\mathrm{AG}]\!]\ [\![\text{nobody}]\!]$ is replaced by $[\![\text{nobody}]\!]^*\ [\![\mathrm{AG}]\!]$, where $[\![\text{nobody}]\!]^*$ is a type raised version of $[\![\text{nobody}]\!]$. This leads to more complicated types for NPs.

$\text{EV}'_{\mathbb{T}}$

$$b_{\mathtt{np}^k,s} : [c^1, ..., c^k] : \mathbb{T}$$

$$\begin{aligned} b_{\mathtt{n}} : [v_e] &: \mathbb{T} \\ \mathtt{role}^1_{eet} : [v, c^k_e] &: \mathbb{T} \\ &\vdots \\ \mathtt{role}^k_{eet} : [v, c^{k-1}_e] &: \mathbb{T} \end{aligned}$$

v_e is fresh, $0 < k \leq l$ and the verb b has the argument structure $[\mathtt{role}^1, ..., \mathtt{role}^l]$

The rule demonstrates how neo-Davidsonian event semantics can be accommodated in the tableau system. The antecedent entry represents a lexical verbal term applied to its constant arguments. Roughly speaking, the entry is interpreted as the first-order logic formula $\exists x.b(x, c^1, ..., c^k)$, where $b(x, c^1, ..., c^k)$ is a Davidsonian predicate. The rule introduces a fresh event entity v_e for which the property $b_{\mathtt{n}}$ is asserted, where $b_{\mathtt{n}}$ is a nominalization of $b_{\mathtt{np}^k,\mathtt{s}}$. For instance, if $b_{\mathtt{np}^k,\mathtt{s}} = \mathbf{kiss}_{\mathtt{np},\mathtt{np},\mathtt{s}}$, then the term $\mathbf{kiss}_{\mathtt{n}}$ corresponds to the property of "*being a kissing event*".[18] Based on the assumption that the thematic roles associated with a verb are provided by some lexicon or a signature, ($\text{EV}'_{\mathbb{T}}$) links the event entity to the arguments via the thematic relational terms corresponding to those roles. Notice that the last argument c^k is associated with the first role because we assume that the order in the argument list follows standard syntactic analyses: a subject is the last to form a constituent with a VP. In case one of the arguments c^i is of type $\mathtt{np}$, its semantic counterpart c_e is linked to the event entity.

Since ($\text{EV}'_{\mathbb{T}}$) decomposes the antecedent into a conjunction of $k+1$ entries, the counterpart rule ($\text{EV}'_{\mathbb{F}}$), which applies to the entry with the false sign, will have $k+1$ branches. Each of the branches will contain exactly one consequent of ($\text{EV}'_{\mathbb{T}}$) with the false sign. Moreover, the rule will be a so-called *consumer* (§ 5.2.1), i.e. a γ-rule, which means that it becomes applicable every time a fresh entity constant is introduced on the branch. Due to several branches and its consumer nature, ($\text{EV}'_{\mathbb{F}}$) is extremely inefficient from theorem proving perspective: for transitive verbs it is a γ-rule with three branches.

The solution with ($\text{EV}'_{\mathbb{T}}$), which accommodates event semantics in LLFs, leads to correct event quantification for the following reasons. (a) The event entity is introduced from the verb if and only if the verb occurs in a true context (i.e. with the true sign). In this way, no event entity is introduced for (17e) with $\mathbb{T}$. (b) The event quantifier has the narrowest scope as the event entity is always introduced from the entry that has a lexical verb and the constant arguments.

In general, a verb can be modified by finitely many adverbials. On the other hand, ($\text{EV}'_{\mathbb{T}}$) introduces an event entity from the true entry with a verb as its main LLF. This means that first we have to strip adverbials away from the head verb and then apply the rule that introduces an event entity. But how can we retrieve discarded adverbials back in order to modify the introduced event entity? It is crucial to save the discarded adverbials

[18] Notice that the term $\mathbf{jog}_{\mathtt{n}}$ differs from $\mathbf{jog}_{\mathtt{vp}}$. While $\mathbf{jog}_{\mathtt{n}}$ encodes the property of being a jogging event, $\mathbf{jog}_{\mathtt{vp}}$ is understood as the property of being jogging. In Chapter 3, we obtain LLFs where each lexical term will be accompanied with a POS tag from https://www.cis.upenn.edu/~treebank/tokenization.html. Therefore, $\mathbf{jog}_{\mathtt{n}}$ and $\mathbf{jog}_{\mathtt{vp}}$ are actually abbreviations of the terms $\mathbf{jog}^{\mathtt{NN}}_{\mathtt{n}}$ and $\mathbf{jog}^{\mathtt{VB}}_{\mathtt{vp}}$, respectively. Hence their corresponding terms of type et, namely $\mathtt{jog}^{\mathtt{NN}}_{et}$ and $\mathtt{jog}^{\mathtt{VB}}_{et}$, are also different terms.

somewhere and retrieve it back when needed. For this reason, we introduce an extra slot, called a *memory* or *modifier list*, in a tableau entry and as a result we get a new ternary format:

$$\underbrace{\texttt{memoryList} : \texttt{LLF} : \texttt{argumentList}}_{\text{ternary format of a term}} : \texttt{truthSign} \tag{18}$$

Similarly to the argument list, which keeps argument terms, the memory list can be seen as keeping function terms. From the linguistic perspective, the parts of the *ternary format* can be seen as trichotomy between modifiers, a head and arguments. Concerning event semantics, we will use the memory list as a storage for the adverbials that indirectly modify a verb. Empty memory lists will be conventionally omitted from tableau entries.

In order to reduce LLFs to the ternary format, we introduce the *modifier pushing* (M>) and *modifier pulling* (M<) rules. Notice that the orientation of the less-than sign shows the direction the term is carried in. The rules can operate on any function term, but we call it modifier rules as they are mainly to shift modifiers.

M<

$$[\vec{M}] : A\ H : [\vec{C}] : \mathbb{X}$$
$$[\vec{M}, A] : H : [\vec{C}] : \mathbb{X}$$

M>

$$[\vec{M}, A] : H : [\vec{C}] : \mathbb{X}$$
$$[\vec{M}] : A\ H : [\vec{C}] : \mathbb{X}$$

The rules are the analogy of the argument pulling (A<) and pushing (A>) rules from §2.1: they carry modifier terms back and forth to a memory list. The following proposition holds for the ternary format and simply typed λ-terms:

Proposition 1. *Using the rules (A<), (A>), (M<) and (M<), a typed term can be reduced to several ternary formats and from each of these ternary formats exactly one term is obtainable, the initial term.*

Proof. Consider a term in one of the ternary forms $[\vec{M}, m] : H : [c, \vec{C}]$ where two possibilities for reconstruction can arise. Due to typed terms, there is only one scenario of putting the terms back to H: either we can pull only c or only m. □

For example, the term in (16e) is possible to be reduced to (16e′) and (16e″) while from them we can recover only one LLF, which is obviously the original term (16e). In other words, Proposition 1 says that it is safe to carry out the pulling and pushing rules as the semantics of entries are not changed by these rules.

$$[\textbf{at\_midnight}_{\text{vp,vp}}, \textbf{in\_Tilburg}_{\text{vp,vp}}, \textbf{slowly}_{\text{vp,vp}}] : \textbf{jog}_{\text{vp}} : [\textbf{John}_{\text{np}}] \tag{16e′}$$
$$[\textbf{at\_midnight}_{\text{vp,vp}}] : \textbf{in\_Tilburg}_{\text{vp,vp}} : [\textbf{slowly}_{\text{vp,vp}}\textbf{jog}_{\text{vp}}, \textbf{John}_{\text{np}}] \tag{16e″}$$

For the semantic analysis, we are interested in the ternary forms like (16e′) as the trichotomy is relevant from the semantic or syntactic viewpoints. Now it seems intuitive how to get the event semantics with proper event modification from an LLF in the ternary format. The event introducing rule ($\text{EV}'_{\mathbb{T}}$) has to take into account the modifier terms in the memory list of an antecedent and assert them for the introduced event entity. To demonstrate this, we upgrade ($\text{EV}'_{\mathbb{T}}$) to ($\text{EV}_{\mathbb{T}}$), which now handles the memory list:

$\text{EV}_{\mathbb{T}}$

$$[M^1_{\texttt{vp},\texttt{vp}}, ..., M^j_{\texttt{vp},\texttt{vp}}] : b_{\texttt{np}^k,\texttt{s}} : [c^1, ..., c^k] : \mathbb{T}$$

$$[A^1_{\texttt{n},\texttt{n}}, ..., A^j_{\texttt{n},\texttt{n}}] : b_{\texttt{n}} : [v_e] : \mathbb{T}$$
$$\texttt{role}^1_{eet} : [v, c^1_e] : \mathbb{T}$$
$$\vdots$$
$$\texttt{role}^k_{eet} : [v, c^k_e] : \mathbb{T}$$

v_e is fresh, $0 < k \leq l$, the verb b has the argument structure $[\texttt{role}^1, ..., \texttt{role}^l]$, for $i = 1, ..., j$
$M^i_{\texttt{vp},\texttt{vp}} = p^i_{\texttt{np},\texttt{vp},\texttt{vp}} H^i$ or $M^i_{\texttt{vp},\texttt{vp}}$ is lexical and
$A^i_{\texttt{n},\texttt{n}} = p^i_{\texttt{np},\texttt{n},\texttt{n}} H^i$ or $A^i_{\texttt{n},\texttt{n}} = der(\texttt{JJ}, M^i_{\texttt{n},\texttt{n}})$, respectively

Similarly to ($\text{EV}'_{\mathbb{T}}$), the rule ($\text{EV}_{\mathbb{T}}$) introduces $b_{\texttt{n}}$, the nominalization of the verb term $b_{\texttt{np}^k,\texttt{s}}$, and asserts it for the event entity. It also introduces the memory list with the nominal modifiers $[A^i_{\texttt{n},\texttt{n}}]^j_{i=1}$ that corresponds to the list of adverbials $[M^i_{\texttt{vp},\texttt{vp}}]^j_{i=1}$. Here, we assume that the adverbials in the memory list are either lexical terms or PPs. A nominal modifier is obtained from an adverbial PP by changing the type $(\texttt{np}, \texttt{vp}, \texttt{vp})$ of a preposition with $(\texttt{np}, \texttt{n}, \texttt{n})$. To obtain an adjective from a lexical adverb, we assume a derivation function der.[19] The consequent entry with $b_{\texttt{n}}$ and the memory list is further analyzed by additional rules for modifier-nominal pairs (presented in § 2.4.1). The usage of the newly introduced rules and the memory list is illustrated by the tableaux in Figure 2.4 and Figure 2.5. The empty memory list is conventionally omitted from the entries.

We have extended the format of tableau entries with the additional list, called a memory or modifier list. The extension is conservative and backward compatible as the old format can be obtained by having the empty memory list. The memory list facilitates linking a remote modifier to its head. The new rules (M<) and (M>) enable to save and discharge terms from the list. The rule for events ($\text{EV}_{\mathbb{T}}$) employs the memory list and reduces the analysis of modifier-VP pairs to the analysis of modifier-noun pairs. In the next section, we will present the additional rules that operate on the elements of a memory list.

2.4 Extending the inventory of rules

An inventory of tableau rules represents a reasoning power for tableau systems. In order to make the natural tableau more powerful and robust, we add additional rules to the inventory. In the previous sections, we have presented the tableau rules of a formal nature, so-called *formal rules*. They are abstract in the sense that they do not depend much on concrete lexical entries. The formal rules usually model general syntactic constructions, algebraic properties, and formal properties licensed by the λ-calculus and the format of tableau entries. For example, (M>) and (M<) are examples of formal rules. In this section, we continue introducing formal rules. In particular, we present several rules concerning the new format of tableau entries and the algebraic properties of modifiers such as *subsectivity* and *commutativity*. We also extend the formal language of TY_{SS} with the exclusion

[19] The function der is a partial function. It takes a lexical term $t^{\texttt{POS1}}$ and a POS tag POS2; if there exists the lexical term $u^{\texttt{POS2}}$ that is a derivational form of $t^{\texttt{POS1}}$, then it returns $u^{\texttt{POS2}}$; otherwise it is not defined for the input. For example, $der(\texttt{JJ}, \textbf{slowly}^{\texttt{RB}}) = \textbf{slow}^{\texttt{JJ}}$ while $der(\texttt{IN}, \textbf{slowly}^{\texttt{RB}})$ is undefined.

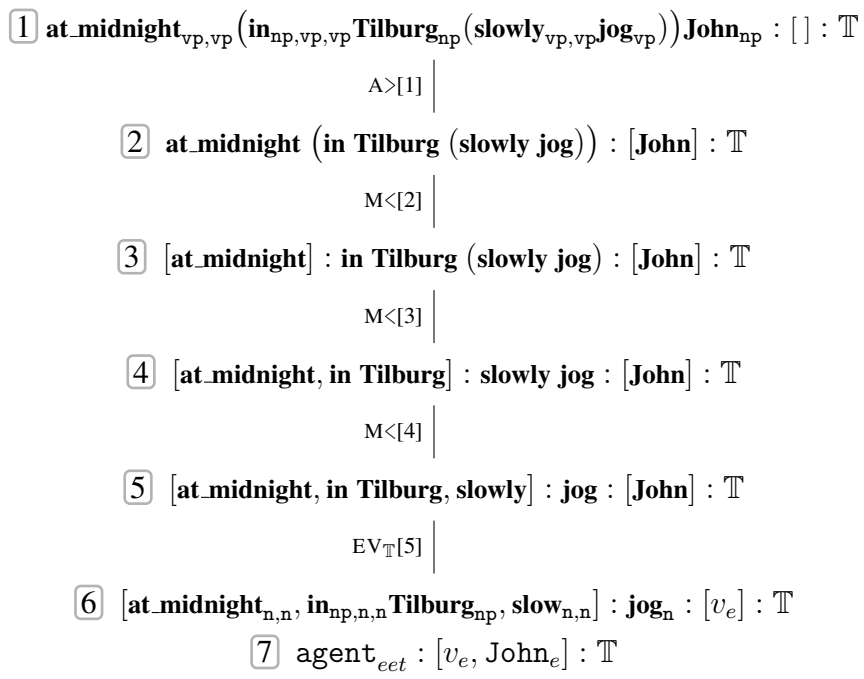


Figure 2.4: The tableau demonstrates the usage of (M<) and (EV$_{\mathbb{T}}$) and shows how they unfold the semantics of (16e). For better demonstration of (EV$_{\mathbb{T}}$), we treat **in_Tilburg**$_{vp,vp}$ as a compound term. It is assumed that **jog**$_{vp}$ has the argument structure [agent].

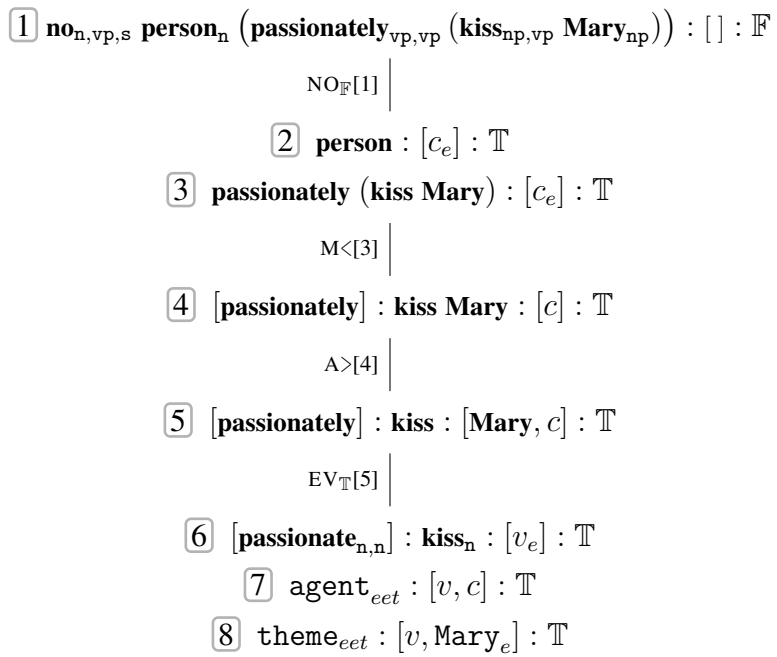


Figure 2.5: The tableau shows how (M<) and (EV$_{\mathbb{T}}$) contribute to decompose the semantics of negated (17e). For transparency and simplicity we substitute **nobody**$_{vp,s}$ with **no**$_{n,vp,s}$ **person**$_{n}$. It is assumed that **kiss**$_{vp}$ has the argument structure [agent, theme].

($|$) and exhaustion ($\smile$) relations since they are as crucial for natural reasoning as the subsumption/inclusion relation ($\sqsubseteq$). The rules modeling various aspects of these relations are presented in the end of the section.

2.4.1 Rules for modifiers and the memory list

In §2.3, we have already presented the rules (M>) and (M<), which deal with the memory list. It has been also shown how to reduce adverbial-verb constructions to the modifier-noun constructions. Following this line, we plan to present the rules that process the modifiers in a memory list. Before doing so, we introduce several algebraic properties, like *subsectivity* and *intersectivity*, of modifier terms, i.e. terms that are of type (α, α). While doing so, we consider only compositional modifier-head constructions; therefore, the phrases like "*hot dog*", "*green thumb*" or "*red tape*" with their non-compositional meanings are ignored.

Definition 12 (Subsectivity and intersectivity)**.** Given a modifier term $A_{\alpha,\alpha}$, where α is a relational type, we say that $A_{\alpha,\alpha}$ is *subsective*, written as $A^{\subset}$, if and only if it satisfies:

$$\forall X(A_{\alpha,\alpha}X_\alpha \sqsubseteq X_\alpha)$$

A modifier term that is not subsective is *non-subsective*. We say that $A_{\alpha,\alpha}$ is *intersective*, written as $A^{\cap}$, if and only if it satisfies:

$$\exists A'\forall X(A_{\alpha,\alpha}X_\alpha = A'_\alpha \sqcap X_\alpha)$$

A modifier term that is not intersective is *non-intersective*.

Corollary 1. Every intersective term is subsective.[20]

Proof. Follows from the fact that $\forall X\forall Y(X_\alpha \sqcap Y_\alpha \sqsubseteq Y_\alpha)$. □

Corollary 2. Every intersective term is upward monotone in its first argument position.

Proof. Follows from $\forall XYF\big((X_\alpha \sqsubseteq Y_\alpha) \rightarrow ((F_\alpha \sqcap X_\alpha) \sqsubseteq (F_\alpha \sqcap Y_\alpha))\big)$, where the latter is licensed by the upward monotonicity of the infimum $\inf$ operator defined on the Boolean algebra $\{0, 1\}$. □

Based on the definitions, we consider the terms **round**$_{\text{n,n}}$, **red**$_{\text{n,n}}$, **plastic**$_{\text{n,n}}$ and **Dutch**$_{\text{n,n}}$ as intersective, hence subsective too. The terms that are subsective and non-intersective are **tall**$_{\text{n,n}}$, **skillful**$_{\text{n,n}}$, **successful**$_{\text{n,n}}$ and **slow**$_{\text{n,n}}$. This is because a "*tall grass*" is not tall in general but tall relatively to grass, a "*skillful surgeon*" is not skillful in everything but in surgery, a "*successful sportsman*" is successful in sports and not necessarily in any other activity, a "*slow jog*" is not slow event in general since it is fast in terms of a movement. On the other hand, the terms **possible**$_{\text{n,n}}$, **apparent**$_{\text{n,n}}$ and **former** are non-subsective modifiers: not every "*possible mistake*" or "*apparent mistake*" is a "*mistake*" and a "*former student*" is not a "*student*". We will come back to these properties when presenting the tableau rules for modifiers in §4.1.2.

[20] Due to this relation, sometimes subsective adjectives are referred as *non-intersective but subsective* in order to distinguish them from intersective adjectives, which are also subsective.

For the semantic perspective, sometimes the order in which modifiers modify a head is irrelevant. Though for syntax it might matter. For instance, English has a preference to a certain order of modifiers on the surface level. For instance, "*round blue toy*" is preferred to "*blue round toy*", or "*relational modifier term*" to "*modifier relational term*". But in both cases, the order of modifiers are usually semantically irrelevant as we are talking about a "*round and blue toy*" or a "*relational and modifier term*", respectively. In order to capture the irrelevance of the order of modifiers, we define the notion of commutativity:

Definition 13 (Commutativity)**.** We say that relational modifier terms $A_{\alpha,\alpha}$ and $B_{\alpha,\alpha}$ *commute with respect to* N_α and define it as:

$$A \rightleftharpoons_N B \stackrel{\text{def}}{=} (A\,B\,N = B\,A\,N)$$

We say that $A_{\alpha,\alpha}$ and $B_{\alpha,\alpha}$ *commute* and define it as:

$$A \rightleftharpoons B \stackrel{\text{def}}{=} \forall X_\alpha (A \rightleftharpoons_X B)$$

Corollary 3. Every two intersective terms commute.

Proof. Follows from the free order of elements in the argument of the infimum $\inf$ operator defined on the Boolean algebra $\{0, 1\}$. □

For example, the following terms commute $\textbf{blue}_{\text{n,n}} \rightleftharpoons \textbf{round}_{\text{n,n}}$ as they are intersective. An example of a pair of subsective terms that are non-intersective but still commute is: $\textbf{tall}_{\text{n,n}} \rightleftharpoons \textbf{skullful}_{\text{n,n}}$. Additionally taking into account that $\textbf{tall}_{\text{n,n}}$ is subsective, we get the following relations (informally written as a chain):

$$\textbf{tall}_{\text{n,n}}\textbf{skullful}_{\text{n,n}}\textbf{surgeon}_{\text{n}} = \textbf{skullful}_{\text{n,n}}\textbf{tall}_{\text{n,n}}\textbf{surgeon}_{\text{n}} \sqsubseteq \textbf{tall}_{\text{n,n}}\textbf{surgeon}_{\text{n}} \tag{19}$$

In this way, commutativity helps to capture direct modification of the head $\textbf{surgeon}_{\text{n}}$ by the remote modifier $\textbf{tall}_{\text{n,n}}$. Obviously not all modifiers commute, e.g., $\textbf{former}_{\text{n,n}}$ and $\textbf{successful}_{\text{n,n}}$ do not commute as a "*former successful student*" is not necessarily a "*successful former student*".

After defining the relevant properties of modifiers, we present the rules that operate on the modifiers of a memory list. The first rule, ($\rightleftharpoons$M), swaps the adjacent terms in the memory list that commute. Since terms commute, this means that the terms are modifiers of the same type. The rule ($\subset \text{M}_{\mathbb{T}}$) concerns subsective modifiers. If the initial term in the list is a subsective modifier, marked with $\subset$ in a superscript, then discarding it preserves the truth. The soundness of ($\rightleftharpoons$M) and ($\subset \text{M}_{\mathbb{T}}$) is obvious if we go from the ternary format back to LLFs and use the definitions for commutative and subsective terms.

$\rightleftharpoons$M

$$\frac{[\vec{A}, M, M', \vec{B}] : H : [\vec{C}] : \mathbb{X}}{[\vec{A}, M', M, \vec{B}] : H : [\vec{C}] : \mathbb{X}}\; M \rightleftharpoons M'$$

$\subset \text{M}_{\mathbb{T}}$

$$\frac{[M^{\subset}, \vec{A}] : H : [\vec{C}] : \mathbb{T}}{[\vec{A}] : H : [\vec{C}] : \mathbb{T}}$$

The rules ($\rightleftharpoons$ M) and ($\subset \mathrm{M}_{\mathbb{T}}$) are fundamental rules and their combinations can lead to robust and linguistically relevant inference rules. One of such rules is ($\Subset_{\mathbb{T}}$). If the modifier M is surrounded by other subsective modifiers and commutes with all modifiers in the path to an H, then applying M directly to H upholds the truth. To show that a list comprises only subsective terms, we mark it with $\subset$ in a superscript. The rule has a gray background to indicate that it is a combination of other rules. The tableaux in Figure 2.8 (Appendix A) demonstrate how ($\Subset_{\mathbb{T}}$) processes nominal and verbal modifications uniformly.

$\Subset_{\mathbb{T}}$

$$[\vec{A}^{\subset}, M, \vec{B}^{\subset}] : H : [\vec{C}] : \mathbb{T}$$
$$MH : [\vec{C}] : \mathbb{T}$$
$$\forall E \in \vec{B}\ (M \rightleftharpoons E)$$

$\times$M

$$[\vec{M}] : A : [\vec{C}] : \mathbb{T}$$
$$[\vec{M}] : B : [\vec{C}] : \mathbb{F}$$
$$\times$$
$$A = B, \text{ or } A \sqsubseteq B \text{ and } \vec{M}^{\uparrow}$$

The tableau in Figure 2.8b serves as an evidence that, with the help of the memory list and the introduced rules, it is not necessary to appeal for event semantics in order to model optionality or disposition of verbal modification. In particular, with the new tableau rules, it is possible to capture the equivalence of (16e) and (16f)[21] and the entailments of (16g) from (16e) or (16f) without introducing event entities.

$$\exists e\big(\texttt{jog}(e) \wedge \texttt{AGENT}(\texttt{John}, e) \wedge \texttt{slow}(e) \wedge \texttt{in\_Tilburg}(e) \wedge \texttt{at\_midnight}(e)\big) \quad (16b)$$
$$\textbf{at\_midnight}_{\mathrm{vp,vp}}\big(\textbf{in\_Tilburg}_{\mathrm{vp,vp}}\big(\textbf{slowly}_{\mathrm{vp,vp}}\ \textbf{jog}_{\mathrm{vp}}\big)\big)\textbf{John}_{\mathrm{np}} \quad (16e)$$
$$\textbf{in\_Tilburg}_{\mathrm{vp,vp}}\big(\textbf{at\_midnight}_{\mathrm{vp,vp}}\big(\textbf{slowly}_{\mathrm{vp,vp}}\ \textbf{jog}_{\mathrm{vp}}\big)\big)\textbf{John}_{\mathrm{np}} \quad (16f)$$
$$\textbf{in\_Tilburg}_{\mathrm{vp,vp}}\big(\textbf{at\_midnight}_{\mathrm{vp,vp}}\ \textbf{jog}_{\mathrm{vp}}\big)\textbf{John}_{\mathrm{np}} \quad (16g)$$

Another useful shortcut rule is ($\times$M). The rule is more general than the closure rule ($\times\sqsubseteq$) as it allows nonempty memory lists in entries. In case A is subsumed by B, all the members of $\vec{M}$ are required to be upward monotone. Including ($\times$M) in the inventory contributes to shorter tableau proofs. Without it, rendering the antecedent nodes inconsistent requires several applications of ($\uparrow\sqsubseteq$). The analogous closure rule can be designed when the elements in the memory lists are downward monotone.

With the help of ($\times$M), if adverbials are upward monotone, we can substitute the verb with a more general verb and hence license the entailment of (16g′) from (16g). The approaches that cannot model subsective adjectives allow wrong entailments like "*John jogged slowly at midnight in Tilburg*" $\models$ "*John moved slowly at midnight in Tilburg*" (triggered by the unsound entailment "*slow jogging*" $\models$ "*slow moving*"). For example, the (neo-) Davidsonian analysis with first-order logic wrongly proves these entailments as it entails (16b′) from (16b). On the other hand, such wrong entailments are blocked in the tableau system as it accounts subsective operators properly: it is not possible to prove (16f) $\vdash$ (16f′) with the current tableau rules.

$$\exists e\big(\texttt{move}(e) \wedge \texttt{AGENT}(\texttt{John}, e) \wedge \texttt{slow}(e) \wedge \texttt{in\_Tilburg}(e) \wedge \texttt{at\_midnight}(e)\big) \quad (16b')$$

[21] Using (M>) and (M<), first reduce both LLFs to the form where all the adverbials and verbal arguments are in the corresponding lists. Then swap $\textbf{at\_midnight}_{\mathrm{vp,vp}}$ and $\textbf{in\_Tilburg}_{\mathrm{vp,vp}}$ via ($\rightleftharpoons$ M). Finally, we can get two identical entries with opposite truth signs by pushing all members of the memory lists.

$$\textbf{in\_Tilburg}_{\text{vp,vp}}\big(\textbf{at\_midnight}_{\text{vp,vp}}\big(\textbf{slowly}_{\text{vp,vp}}\ \textbf{move}_{\text{vp}}\big)\big)\textbf{John}_{\text{np}} \quad (16f')$$

$$\textbf{in\_Tilburg}_{\text{vp,vp}}\big(\textbf{at\_midnight}_{\text{vp,vp}}\ \textbf{move}_{\text{vp}}\big)\textbf{John}_{\text{np}} \quad (16g')$$

It is noteworthy that the rules ($\not\in_{\mathbb{T}}$) and ($\times$M) are admissible rules, meaning that they are redundant from the completeness point of view.[22] For instance, ($\not\in_{\mathbb{T}}$) can be represented by the several applications of ($\rightleftharpoons$M), ($\subset \text{M}_{\mathbb{T}}$) and (M$<$). Conventionally we display admissible rules with a gray background.

The new rules have been introduced which process a memory list depending on the algebraic properties of its modifiers. The analogy between the ternary format of entries and the syntactic modifier-head-argument distinction in natural languages seems promising from the natural logic perspective. As an evidence, modeling the phenomena like optionality and disposition of adverbial modifiers do not require deep semantic processing, e.g., an introduction of event entities. Nevertheless, event semantics and the rule (EV$_{\mathbb{T}}$) are still indispensable when dealing with examples like the paraphrases concerning the argument structure of a verb or reasoning over temporal references.

2.4.2 Rules for semantic exclusion and exhaustion

The semantic inclusion (or subsumption) relation $\sqsubseteq$ is crucial for natural reasoning. It underlies one of the most fundamental tableau rules ($\times\sqsubseteq$) which closes tableau branches. In addition to $\sqsubseteq$, here we incorporate two new semantic relations, *exclusion* and *joint exhaustion*, which are also crucial for the reasoning. The tableau rules related to these relations are presented and shown how they contribute to proofs of certain logical relations.

The semantic inclusion relation is well studied in terms of monotonicity calculus since van Benthem (1986); Sánchez-Valencia (1991). Later, MacCartney (2009) and MacCartney and Manning (2009), along with monotonicity calculus, incorporated additional relations into natural logic including exclusion and exhaustion. From the natural logic perspective, this step seems quite intuitive as the inclusion relation with monotonicity calculus is just a part of a larger calculus (Icard, 2012; Icard and Moss, 2014). We follow MacCartney (2009) and introduce additional relations in $\texttt{TY}_{\texttt{SS}}$.[23]

Definition 14 (Exclusion)**.** We say that relational terms $A_{\vec{\alpha}t}$ and $B_{\vec{\alpha}t}$ are in an *exclusion* relation (or are *disjoint*) and define it as the formula of $\texttt{TY}_{\texttt{SS}}$:

$$A \mid B \stackrel{\text{def}}{=} \neg\exists\vec{X}.\,(A \sqcap B)\vec{X}$$

Definition 15 (Joint exhaustion)**.** We say that relational terms $A_{\vec{\alpha}t}$ and $B_{\vec{\alpha}t}$ are *jointly exhaustive* (or represent a *cover*) and define it as the formula of $\texttt{TY}_{\texttt{SS}}$:

$$A \smile B \stackrel{\text{def}}{=} \forall\vec{X}.\,(A \sqcup B)\vec{X}$$

[22] For more discussion about redundant tableau rules see §5.2.1.

[23] In contrast to MacCartney (2009), we do not adopt the most specific relations but the relations that are based on more general properties. In particular, MacCartney (2009) interprets $x|y$ as a conjunction of $x \cap y = \emptyset$ and $x \cup y \neq U$, while $x \smile y$ is understood as a conjunction of $x \cap y \neq \emptyset$ and $x \cup y = U$, where U is a universal set. In our case, similarly to Icard (2012); Icard and Moss (2014), $x|y$ will mean only $x \cap y = \emptyset$ and $x \smile y$ only $x \cup y = U$. The symbols for the exclusion ($|$) and exhaustion ($\smile$) relations are borrowed from MacCartney and Manning (2009).

Notice that the latter two definitions can be equivalently expressed syntactically as $A \mid B \stackrel{\text{def}}{=} A \sqsubseteq -B$ and $A \smile B \stackrel{\text{def}}{=} -A \sqsubseteq B$, respectively. The semantics of the defined terms can be shortly described as follows: $[\![A \mid B]\!]^{\mathcal{M},\mathfrak{a}} = 1$ iff $[\![A \sqcap B]\!]^{\mathcal{M},\mathfrak{a}} = 0_{\vec{\alpha}t}$ and $[\![A \smile B]\!]^{\mathcal{M},\mathfrak{a}} = 1$ iff $[\![A \sqcup B]\!]^{\mathcal{M},\mathfrak{a}} = 1_{\vec{\alpha}t}$. The exclusion relation is more general than the antonymy relation. Since many concepts in natural language are specific, disjoint concepts are frequent. The examples of the lexical terms in the exclusion relation are:

$$(\textbf{dog}_{\text{n}} \mid \textbf{cat}_{\text{n}}), \quad (\textbf{many}_{\text{n,vp,s}} \mid \textbf{few}_{\text{n,vp,s}}), \quad (\textbf{sleep}_{\text{vp}} \mid \textbf{run}_{\text{vp}})$$
$$(\textbf{cheap}_{\text{n,n}} \mid \textbf{expansive}_{\text{n,n}}) \quad \text{and} \quad (\textbf{with}_{\text{np,n,n}} \mid \textbf{without}_{\text{np,n,n}})$$

In contrast to disjoint concepts, natural language does not offer so many jointly exhaustive pairs of lexical words. The following pairs of terms (including both lexical and compound) are jointly exhaustive:

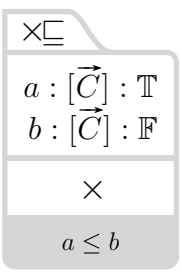
$$(\textbf{nonhuman}_{\text{n,n}} \smile \textbf{animal}_{\text{n,n}}), \quad (\textbf{at least six}_{\text{n,vp,s}} \smile \textbf{at most ten}_{\text{n,vp,s}}) \quad \text{and} \quad (\textbf{some}_{\text{n,vp,s}} \smile \textbf{no}_{\text{n,vp,s}})$$

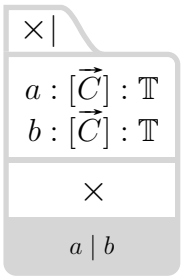
The pairs of terms that are in both exclusion and exhaustion relations are:

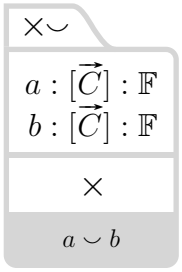
$$\langle \textbf{human}_{\text{n}}, \textbf{nonhuman}_{\text{n}} \rangle, \quad \langle \textbf{some}_{\text{n,vp,s}}, \textbf{no}_{\text{n,vp,s}} \rangle \quad \text{and} \quad \langle \textbf{with}_{\text{np,n,n}}, \textbf{without}_{\text{np,n,n}} \rangle$$

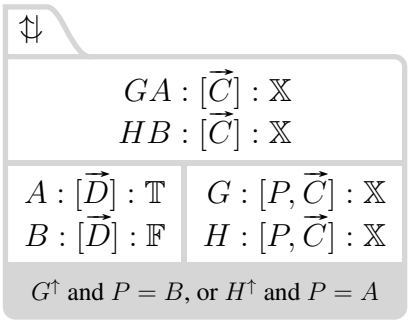
We call such pairs of terms *complementary* or *contradictory*. The semantic relations over lexical terms are assumed to be provided by some KB.

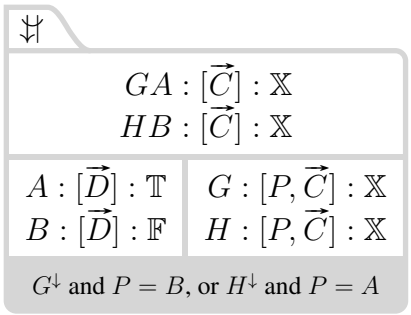
The exclusion and exhaustion relations over lexical terms are modeled by the ($\times\mid$) and ($\times\smile$) tableau rules. They apply to the entries with the same sign, in contrast to ($\times\sqsubseteq$). The rules together with ($\times\sqsubseteq$) offer a general solution: if a pair of terms is in several relations, then each rule accounts for its corresponding relation. For instance, while proving that two terms are complementary to each other, one uses both ($\times\mid$) and ($\times\smile$) rules. These rules are enough to model all 7 relations used by MacCartney and Manning (2009).











In order to model exclusion and exhaustion over compound LLFs, we need additional rules for that. The rules ($\uparrow\sqsubseteq$) and ($\downarrow\sqsubseteq$) try to align arguments of monotonic functions while hoping that the terms are in a certain inclusion relation. The analogous rules for exclusion and exhaustion are ($\leftrightarrows$) and ($\leftrightarrows$). For instance, ($\leftrightarrows$) processes the entries which

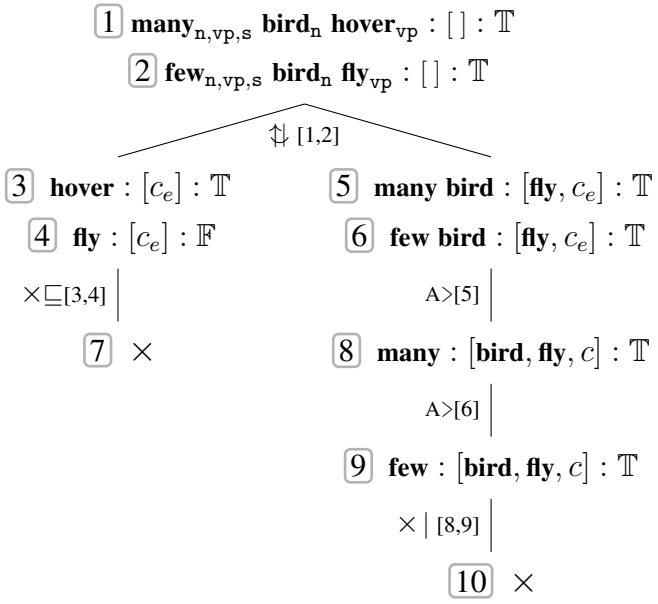


Figure 2.6: The tableau proves that "*many birds hover*" contradicts "*few birds fly*" and vice versa. One of (⇅) and (⇵) is necessary to prove the contradiction.

show that GA and HB are not disjoint (i.e. $\mathbb{X} = \mathbb{T}$) or are not jointly exhaustive (i.e. $\mathbb{X} = \mathbb{F}$). After one of H and G is upward monotone, the rule analyzes the entries as follows. Either $A \not\sqsubseteq B$ in the left branch or $A \sqsubseteq B$ in the right branch, where the latter allows to set the same arguments for G and H. In this way, if $A \sqsubseteq B$ holds, then (⇅) shifts reasoning from GA and HB to the simpler G and H terms. (⇵) does the similar job when one of the function terms is downward monotone. Figure 2.6 illustrates how the new rules contribute to the proof of a contradiction relation that was not possible to be proved before the introduction of the rules.

Certain function terms have properties to modify and project the relations between their arguments. For example, if a relational term $F_{\alpha\vec{\gamma}t}$ is upward (downward) monotone and $X_\alpha \sqsubseteq Y_\alpha$, then $FX \sqsubseteq FY$ ($FY \sqsubseteq FX$ respectively). In other words, upward monotone functions preserve $\sqsubseteq$ while downward monotone functions reverse it. The projectivity behavior of monotone functions can be captured via (F↑⊑) and (F↓⊑).[24] But actually these rules are admissible since (↑⊑) and (↓⊑) can mirror them: when $G = H$ the right consequents of (↑⊑) and (↓⊑) lead to immediate closure while the left ones coincide with the consequents of (F↑⊑) and (F↓⊑) respectively.

F↑⊑

$$\frac{F^{\uparrow}A : [\vec{C}] : \mathbb{T} \quad F^{\uparrow}B : [\vec{C}] : \mathbb{F}}{A : [\vec{D}] : \mathbb{T} \mid B : [\vec{D}] : \mathbb{F}} \; \vec{D} \text{ is fresh}$$

F↓⊑

$$\frac{F^{\downarrow}A : [\vec{C}] : \mathbb{T} \quad F^{\downarrow}B : [\vec{C}] : \mathbb{F}}{A : [\vec{D}] : \mathbb{F} \mid B : [\vec{D}] : \mathbb{T}} \; \vec{D} \text{ is fresh}$$

[24] In order to indicate that the both antecedent entries of the rules share the same function term, we put F in the names of the rules.

Based on the inclusion, exclusion and exhaustion relations, there are 12 different possible projectivity properties for function terms. Two of them, in (9) and (10), represent the projection of inclusion relation expressed by monotone functions. Four projectivity properties of the disjoint and exhaustion relations are given below (the appropriate types of the terms are assumed by default):

$$\forall XY\big((X \mid Y) \to (FX \mid FY)\big) \tag{P1}$$
$$\forall XY\big((X \mid Y) \to (FX \smile FY)\big) \tag{P2}$$
$$\forall XY\big((X \smile Y) \to (FX \smile FY)\big) \tag{P3}$$
$$\forall XY\big((X \smile Y) \to (FX \mid FY)\big) \tag{P4}$$

The rest of the properties correspond to projections that map asymmetric relations (e.g., $\sqsubseteq$ and $\sqsupseteq$) to symmetric relations (e.g., $\mid$ and $\smile$) or vice versa. We call these projections *asymmetric* ones. For the demonstration, two of the six asymmetric projections are given in (P5) and (P6). The asymmetric projectivity properties are somewhat unusual. For example, the property in (P5) implies that any disjoint pair of terms are mapped to semantically equivalent terms, or that for any term X we have $F(X) = F(-X)$. On the other hand, (P6) maps arbitrarily small or large terms to a cover. Moreover, according to it, for any X and Y such that $X \sqsubseteq Y$, both $F(X) \smile F(Y)$ and $F(-X) \smile F(-Y)$ covers hold.

$$\forall XY\big((X \mid Y) \to (FX \sqsubseteq FY)\big) \tag{P5}$$
$$\forall XY\big((X \sqsubseteq Y) \to (FX \smile FY)\big) \tag{P6}$$

While we are not aware of any linguistic phrases that realize asymmetric projections, there are several phrases that satisfy the properties in (P1–P4). Based on these phrases we can say that subsective terms (e.g., $\textbf{skillful}_{\mathtt{n,n}}$), $\textbf{most}_{\mathtt{q}}\, N_{\mathtt{n}}$ and $\textbf{every}_{\mathtt{q}}\, N_{\mathtt{n}}$ satisfy (P1); $\textbf{not}_{\mathtt{q,q}}\, \textbf{every}_{\mathtt{q}}\, N_{\mathtt{n}}$, $\textbf{not}_{\mathtt{vp,vp}}$ and $\textbf{not}_{\mathtt{n,n}}$ satisfy (P2); $\textbf{some}_{\mathtt{q}}\, N_{\mathtt{n}}$ has the property (P3); $\textbf{no}_{\mathtt{q}}\, N_{\mathtt{n}}$, $\textbf{not}_{\mathtt{vp,vp}}$ and $\textbf{not}_{\mathtt{n,n}}$ comply with (P4).[25] The terms containing $N_{\mathtt{n}}$ maintain corresponding projectivity properties only in those models $\mathcal{M}$ where $[\![N_{\mathtt{n}}]\!]^{\mathcal{M}} \neq 0_{\mathtt{n}}$. For example, if this condition is violated, i.e. $[\![N_{\mathtt{n}}]\!]^{\mathcal{M}} = 0_{\mathtt{n}}$, then $\textbf{some}_{\mathtt{q}}\, N_{\mathtt{n}}$ will not satisfy (P3) for $\mathcal{M}$: for a cover ($\textbf{run}_{\mathtt{vp}} \smile \textbf{not}_{\mathtt{vp,vp}}\textbf{run}_{\mathtt{vp}}$), none of $\textbf{some}_{\mathtt{q}}\, N_{\mathtt{n}}\textbf{run}_{\mathtt{vp}}$ and $\textbf{some}_{\mathtt{q}}\, N_{\mathtt{n}}(\textbf{not}_{\mathtt{vp,vp}}\textbf{run}_{\mathtt{vp}})$ are true. Notice that the term $\textbf{most}_{\mathtt{q}}\, N_{\mathtt{n}}$ unconditionally satisfies (P1).

F||	F|(	F((	F(|
$FA : [\vec{C}] : \mathbb{T}$ $FB : [\vec{C}] : \mathbb{T}$	$FA : [\vec{C}] : \mathbb{F}$ $FB : [\vec{C}] : \mathbb{F}$	$FA : [\vec{C}] : \mathbb{F}$ $FB : [\vec{C}] : \mathbb{F}$	$FA : [\vec{C}] : \mathbb{T}$ $FB : [\vec{C}] : \mathbb{T}$
$A : [\vec{D}] : \mathbb{T}$ $B : [\vec{D}] : \mathbb{T}$	$A : [\vec{D}] : \mathbb{T}$ $B : [\vec{D}] : \mathbb{T}$	$A : [\vec{D}] : \mathbb{F}$ $B : [\vec{D}] : \mathbb{F}$	$A : [\vec{D}] : \mathbb{F}$ $B : [\vec{D}] : \mathbb{F}$
$\vec{D}$ is fresh and F satisfies (P1)	$\vec{D}$ is fresh and F satisfies (P2)	$\vec{D}$ is fresh and F satisfies (P3)	$\vec{D}$ is fresh and F satisfies (P4)

The tableau rules that model (P1–P4) properties are presented above. The rules carry out inference from complex terms to simpler ones, where a pair of entries of the form

[25] We assume that $\mathtt{q}$ abbreviates the syntactic type $(\mathtt{n}, \mathtt{vp}, \mathtt{s})$.

$A : [\vec{D}] : \mathbb{X}$ and $B : [\vec{D}] : \mathbb{X}$ amounts to $A \nearrow B$ if $\mathbb{X} = \mathbb{T}$ and $A \not\nearrow B$ if $\mathbb{X} = \mathbb{F}$. The names of the rules follow the intuition that a function maps the first relation to the second one: (F|() expresses the projectivity property that maps the exclusion relation to the exhaustion relation. The presented four rules account for the projectivity signatures studied by MacCartney (2009); Icard (2012).

The properties in (P1–P4) can be seen as an offshoot of the following algebraic properties, studied since (Zwarts, 1981) and (Hoeksema, 1983). The definition follows Icard (2012).

Definition 16. For relational terms $F_{\alpha,\beta}$, A_α and B_α, we say that:

- F is *multiplicative* ($\uparrow_\circ$) iff $F(A \sqcap B) = FA \sqcap FB$, and it is *completely multiplicative* ($\uparrow^c_\circ$) if additionally $[\![F]\!](0_\alpha) = 0_\beta$;
- F is *anti-multiplicative* ($\downarrow_\circ$) if $F(A \sqcap B) = FA \sqcup FB$, and it is *completely anti-multiplicative* ($\downarrow^c_\circ$) if additionally $[\![F]\!](0_\alpha) = 1_\beta$.
- F is *additive* ($\uparrow_+$) iff $F(A \sqcup B) = FA \sqcup FB$, and it is *completely additive* ($\uparrow^c_+$) if additionally $[\![F]\!](1_\alpha) = 1_\beta$;
- F is *anti-additive* ($\downarrow_+$) if $F(A \sqcup B) = FA \sqcap FB$, and it is *completely anti-additive* ($\downarrow^c_+$) if additionally $[\![F]\!](1_\alpha) = 0_\beta$;

For example, $\textbf{no}_{\text{q}}$ is $\downarrow_+$ in both of its arguments, $\textbf{every}_{\text{q}}$ is $\downarrow_+$ in its first argument and $\uparrow_\circ$ in the second argument, $\textbf{some}_{\text{q}}$ is $\uparrow_+$ in both of its arguments, $\textbf{not}_{\text{q,q}}$ $\textbf{every}_{\text{q}}$ is $\uparrow_+$ in its first argument and $\downarrow_\circ$ in the second argument, the negations $\textbf{not}_{\text{vp,vp}}$ and $\textbf{not}_{\text{n,n}}$ are $\downarrow^c_+$ and $\downarrow^c_\circ$ at the same time.[26]

Each defined algebraic properties automatically satisfies exactly one of (P1–P4) properties, preserving the order. The completeness property is necessary for this connection. So, the tableau rules modeling (P1–P4) also automatically work for the terms with the complete versions of the algebraic properties. Moreover, it is not necessary for a term to have one of these four algebraic properties in order to be processed by the tableau rules. For instance, $\textbf{most}_{\text{q}}\ N_{\text{n}}$ is simply upward monotone and does not have non of the four algebraic properties, however it still satisfies (P1) and hence (F||) works for it. In other words, the presented tableau rules model the projectivity properties (P1–P4 which are more basic than the ones in Definition 16.

The section has presented two groups of new tableau rules. The first group consists of the rules operating on the members of a memory list having algebraic properties like subsectivity, intersectivity and commutativity. Several admissible rules were also presented which shorten tableau proofs significantly. The rules of the first group finalize integration of event semantics into the natural tableau system. The second group of rules is dedicated to the exclusion and exhaustion relations. Specifically, we have presented the rules which account for theses relations, their combination with monotonic operators, and certain projectivity behavior of function terms with respect to the relations.

[26] Notice that additivity and multiplicativity automatically assume upward monotonicity, while anit-additivity and anti-multiplicativity imply downward monotonicity. Also multiplicative terms are exactly those terms that are upward monotone and have meet. The terms that are splitting and upward monotone are additive and vice versa.

2.5 Conclusion

We have presented the natural tableau system of Muskens (2010) which is foundation for the work presented in the next chapters. In order to make the presentation accessible for wide range of readers, we have introduced the simple type theory TY and a semantic tableau method and also included several detailed examples of natural tableau proofs. The natural tableau system is prepared for wide-coverage reasoning by extending it in three dimensions: the type system, the format of tableau entries, and the set of tableau rules.

Integration of syntactic types in the type system helps to maintain syntactic information for terms (§ 2.2). Since our goal is to employ natural logic as an inference device, the integration step could be seen as enhancing the logic with syntactic information. This move fits well into the project of natural logic where inference rules and syntactic rules are strongly related to each other (Lakoff, 1970). Interaction between semantic and syntactic terms is established via the subtyping relation: it allows the terms like $\mathbf{red}_{\mathtt{n},\mathtt{n}}\,\mathbf{car}_{\mathtt{n}}\,c_e$ to be well-formed as $\mathtt{n}$ is a subtype of et. Apart from introducing syntactic information, the extension encourages a natural appearance of LLFs. For example, we can express intersective adjectives, e.g., $\mathbf{red}_{\mathtt{n},\mathtt{n}}$ or verbal predicates, e.g., $\mathbf{run}_{\mathtt{np},\mathtt{s}}$, without explicit use of a conjunction and an event entity, respectively. Deeper semantics of such expressions can be later retrieved after tableau rule applications. Finally, syntactic types contribute to fine-grained term matching. The latter is crucial from an automated theorem proving perspective while searching for applicable tableau rules.

We have added an additional slot, in the format of tableau entries, called the *modifier* or *memory* list. Its function is to keep terms and reuse them later if necessary. The list is heavily used by modifiers. Pushing adverbs or remote adjectives in the list is used to establish link with their heads. For example, the modifier list makes it easy to account for event semantics in the natural tableau (§ 2.3.2). An event entity is introduced from a lexical verb if it is signed with $\mathbb{T}$. At the same time, its adverbs are kept in the modifier list and they wait for the introduction of the event entity. After its introduction, the adverbs are asserted for it. Furthermore, the modifier list can be seen as a counterpart of the argument list. In formal logics, completing a predate or a function with its arguments is mainstream while function or predicate modification is allowed only in particular logics. Taking into account the expressive power of natural language, why should not we have a dedicated list for modifiers while we have it for arguments? In the end, the ternary dichotomy modifier-head-argument of a term is another step towards coupling reasoning and syntax in the natural tableau system. Now, in conjunction with the argument list, the modifier list can also be used for aligning identical modifiers and help to single out and contrast heads.

In addition to the extra rules which operate on the modifier list and account for event semantics, we have introduced new rules that model the semantic exclusion and exhaustion relations (§ 2.4.2). The rules can be classified into three groups based on the phenomena they model. The first group contains closure rules that directly account for the two semantic relations. The rules in the second group model interaction between monotonic functions and their arguments which are in one of the two relations. These rules are counterparts of the monotonic rules suggested by Muskens (2010). The third group has rules that model projectivity features of function terms. These rules contrast terms with the same functions and shift reasoning to the arguments.

All in all, the extended tableau system represents a powerful calculus that employs both syntactic and semantic types. The format of tableau entries offers dedicated slots for modifiers, heads and arguments. The information-rich and structured terms are processed with the help of tableau rules. The processing can be of three types: (i) decomposition of a term into smaller parts, (ii) deriving information based on contrasting different terms, and (iii) shifting a term from a shallow level (i.e. with syntactic types) to a deeper level (i.e. with semantic types). After extending and gearing the natural tableau system for wide-coverage reasoning, in the two subsequent chapters, we actually model natural reasoning in the extended system. First, we automatically obtain LLFs from wide range of linguistic expressions (Chapter 3). Then we design a bunch of tableau rules that account for semantics of various lexical elements and syntactic constructions (Chapter 4).

Appendix A

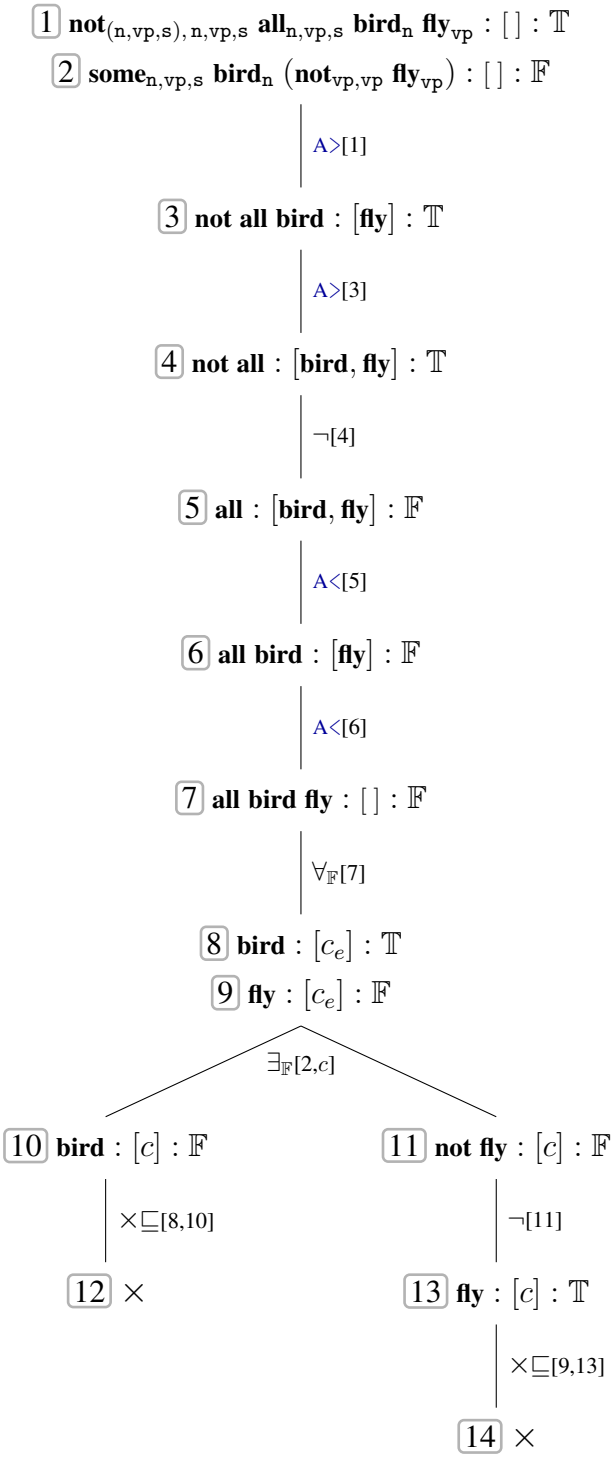
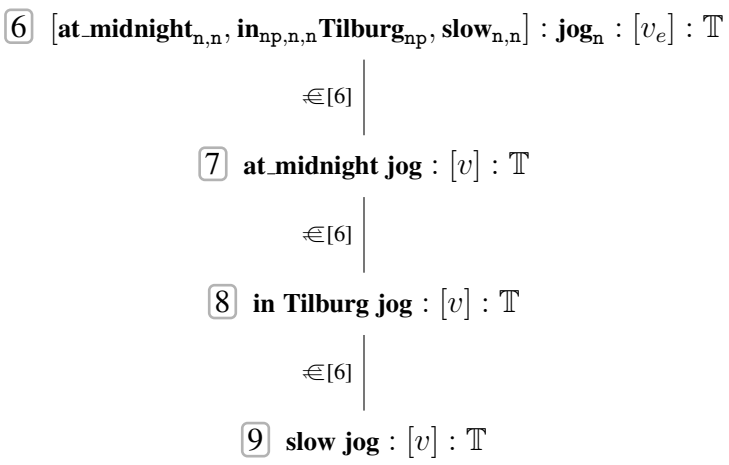
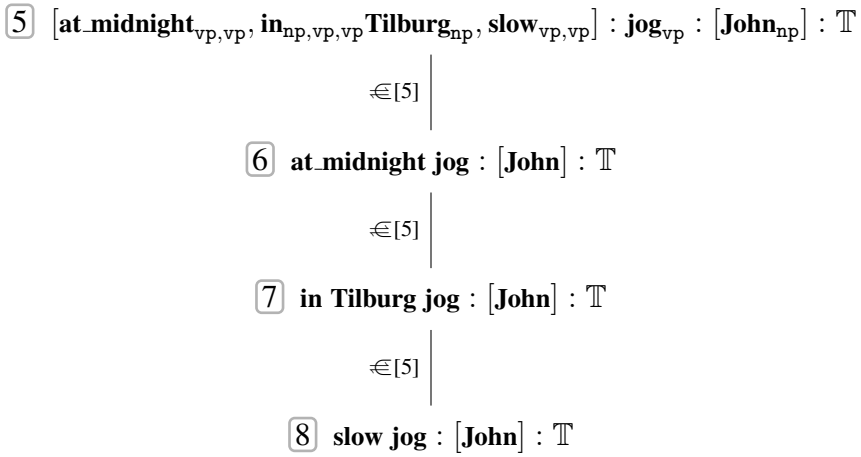


Figure 2.7: The tableau proves that "*not all birds fly*" entails "*some bird does not fly*". Apart from using the TY_{SS} terms, i.e. LLFs with the extended type system, the proof is identical to the tableau found in Figure 2.2.



(a) Processing nominal modification. The modifiers are assumed to commute pairwise and be subsective.



(b) Processing verbal modification. The modifiers are assumed to commute pairwise and be subsective.

Figure 2.8: The tableaux show how the memory list and the rule ($\notin_{\mathbb{T}}$), powered by ($\rightleftharpoons$M) and ($\subset$M$_{\mathbb{T}}$), allow the uniform treatment of nominal (e.g., jog$_{n}$) and verbal (e.g., jog$_{vp}$) modifications. The initial entries 5 and 6 are taken from the tableau in Figure 2.4.

Chapter 3

Lambda Logical Forms for Wide-Coverage Text

In order to enable the natural tableau system to operate on natural language text, it is necessary to generate Lambda Logical Forms (LLFs) automatically from linguistic expressions. In this chapter, we describe how to obtain LLFs from syntactic derivations of categorial grammar. We opt for categorial grammar derivations since there lexical elements are combined like a function and an argument, similarly to the term combination in LLFs. Syntactic derivations will be produced by state-of-the-art parsers that analyze linguistic phrases according to Combinatory Categorial Grammar (CCG). Generating semantically adequate LLFs from CCG derivations is not a trivial procedure. It is challenging for two main reasons: (i) CCG parsers diverged from the CCG formalism in order to process unrestricted text efficiently, and (ii) The parsers are not perfect and therefore they make mistakes. Both the discrepancy from the formalism and the parsers' mistakes need to be eliminated in the generation procedure. The chapter is structured as follows.

We start with introducing the CCG formalism (Steedman, 2000), and then two state-of-the-art CCG parsers, C&C (Clark and Curran, 2007) and EasyCCG (Lewis and Steedman, 2014a), are presented and compared to each other. The procedure of generating LLFs from CCG derivation trees consists of several steps; see Figure 3.1. After obtaining a CCG derivation tree of a phrase, the very first step is to eliminate directional components in the CCG tree, and as a result get, what we call, a CCG term. We carefully analyze CCG trees and draw a list of mistakes, i.e. semantically inadequate analyses, that are systematically made by the CCG parsers. These mistakes in CCG terms are corrected with the help of several hand-written rules. The correction procedure also covers elimination of type-changing rules of the CCG parsers—the rules nonnative for the CCG formalism. Final

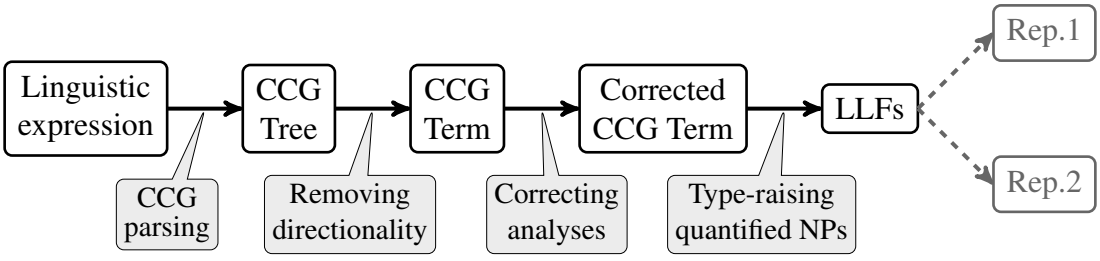


Figure 3.1: A procedure of generating LLFs from a raw linguistic expression.

LLFs are obtained from corrected CCG terms via type-raising quantified noun phrases. The type-raising procedure represents an improved version of the nested Cooper storage (Keller, 1988). This step produces LLFs with scope resolved quantifiers. Implementation details of the generation procedure are also discussed. The implemented LLF generator, called LLFgen, can be seen as a valuable tool for those researcher lines that attempt to obtain linguistic semantics in a compositional manner. With the help of LLFgen, it is possible to get more fine-grained logical forms than one can obtain solely from the CCG parsers. This is due to integration of fixing rules in LLFgen.

3.1 Combinatory Categorial Grammar

Categorial grammar is a lexicalized grammar formalism, in which all constituents are labeled with a *syntactic category* (also often referred as a syntactic type) and a semantic interpretation. A syntactic category can be either a *primitive category* or a *function category*. Usually, N, NP, PP, and S are the primitive categories corresponding to syntactic categories for common nouns, noun phrases (NPs), prepositional phrases (PPs) and sentences, respectively. Depending on applications, the primitive categories can be further distinguished by features, like number, case, mood, etc.

Usually categorial grammars distinguish two directions of functional application which are motivated by word order in linguistic expressions. Consequently function categories have one of the forms Y/X or $Y\backslash X$, where X and Y are meta-variables over syntactic categories. A syntactic category Y/X (or $Y\backslash X$) is a function that returns Y if an argument X is found on its right (or left, respectively). Constituents are then combined using the function application rules: the *forward* ($>$) and *backward* ($<$) application rules feed a function category with an argument category on its right and left, respectively.

$$Y/X : \mathtt{f} \quad X : \mathtt{a} \;\Rightarrow\; Y : \mathtt{fa} \qquad \textit{Forward functional application} \qquad (>)$$
$$X : \mathtt{a} \quad Y\backslash X : \mathtt{f} \;\Rightarrow\; Y : \mathtt{fa} \qquad \textit{Backward functional application} \qquad (<)$$

The rules also come with a recipe how to combine semantic interpretations of constituents, where semantic interpretations are delimited from categories by a semicolon. The interpretations are usually combined using the λ-calculus as it goes hand-in-hand with a function application carried over constituents. The described categorial grammar, with the directional slashes and two functional application rules, is the earliest and simplest categorial grammar known as AB-grammar after Ajdukiewicz (1935) and Bar-Hillel (1953).

Combinatory Categorial Grammar (CCG) is a kind of categorial grammar which, in addition to ($>$) and ($<$), employs extra *combinatory rules*, e.g., type-raising and functional composition rules.[1] The *forward type-raising rule* ($>\mathsf{T}$) changes a category into a functional category that takes functions over this category as an argument. The rule is often used for type-raising NP category to $S/(S\backslash NP)$. The backward version of ($>\mathsf{T}$) is ($<\mathsf{T}$).

$$X : \mathtt{a} \;\Rightarrow\; T/(T\backslash X) : \lambda x.\, x\mathtt{a} \qquad \textit{Forward type-raising} \qquad (>\mathsf{T})$$
$$X : \mathtt{a} \;\Rightarrow\; T\backslash(T/X) : \lambda x.\, x\mathtt{a} \qquad \textit{Backward type-raising} \qquad (<\mathsf{T})$$

[1] The rules in CCG are called combinatory rules since they correspond to the simplest of combinators found in Curry and Feys (1958).

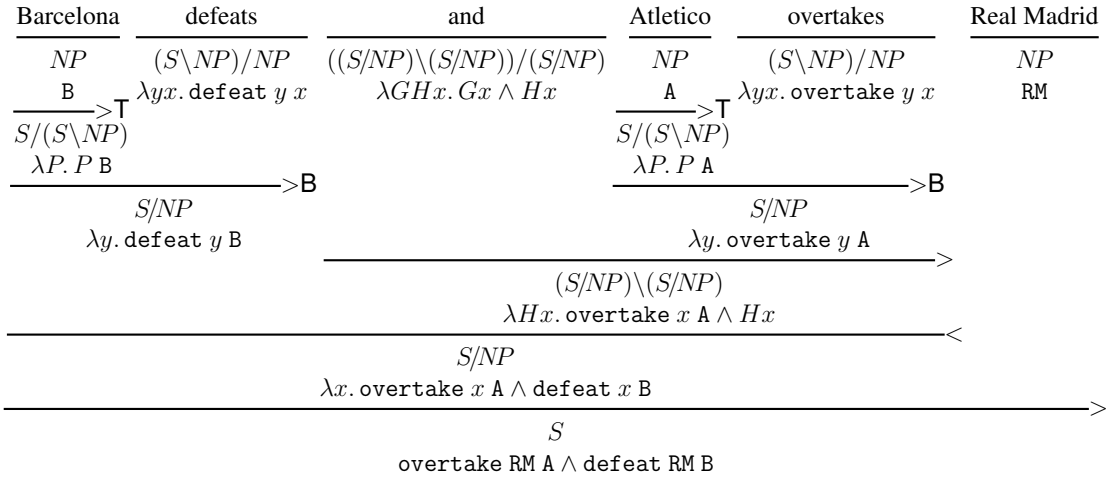


Figure 3.2: A CCG derivation for a non-constituent coordination—right node raising

The composition rules combine two constituents of functional category first, by feeding one of the constituents with a hypothetical constituent, then applying another constituent to the fed one, and in the end eliminating the introduced hypothetical constituent.

$$Z/Y : \mathtt{f} \quad Y/X : \mathtt{g} \;\Rightarrow\; Z/X : \lambda x.\, \mathtt{f}(\mathtt{g}x) \qquad \textit{Forw. functional composition} \qquad (>\mathsf{B})$$

$$Y\backslash X : \mathtt{g} \quad Z\backslash Y : \mathtt{f} \;\Rightarrow\; Z\backslash X : \lambda x.\, \mathtt{f}(\mathtt{g}x) \qquad \textit{Backw. functional composition} \qquad (<\mathsf{B})$$

$$Z/Y : \mathtt{f} \quad Y\backslash X : \mathtt{g} \;\Rightarrow\; Z\backslash X : \lambda x.\, \mathtt{f}(\mathtt{g}x) \qquad \textit{Forw. crossing funct. compos.} \qquad (>\mathsf{B}_\times)$$

$$Y/X : \mathtt{g} \quad Z\backslash Y : \mathtt{f} \;\Rightarrow\; Z/X : \lambda x.\, \mathtt{f}(\mathtt{g}x) \qquad \textit{Backw. crossing funct. compos.} \qquad (<\mathsf{B}_\times)$$

Let us consider the *forward composition rule* ($>\mathsf{B}$) as an example. Since Z/Y cannot take Y/X as an argument or vice versa, first, Y/X applies to a hypothetical argument of category X on its right. The resulted Y is taken as an argument by Z/Y and Z is obtained. Taking away the hypothetical X from the right side of Z results in Z/X. Most common composition rules for English CCG are ($>\mathsf{B}$) and ($<\mathsf{B}_\times$). The other composition rules that might be employed in the CCG framework are crossing counterparts of the former rules, ($>\mathsf{B}_\times$) and ($<\mathsf{B}_\times$), and generalized versions of the composition rules, e.g., ($>\mathsf{B}^2$).

Generalized-2 forward functional composition

$$Z/Y : \mathtt{f} \quad (Y/X)/W : \mathtt{g} \;\Rightarrow\; (Z/X)/W : \lambda wx.\, \mathtt{f}(\mathtt{g}wx) \qquad (>\mathsf{B}^2)$$

In order to demonstrate how the CCG combinatory rules work, a non-trivial example of a CCG derivation is given in Figure 3.2. The derivation employs ($>$, $<$, $>\mathsf{T}$, $>\mathsf{B}$) rules and shows how syntactic categories and semantics of phrases are obtained from syntactic categories and semantics of its constituents. In the end, the sentence is of category S with a first-order logic (FOL) formula as its semantic interpretation.[2]

The CCG formalism, like other categorial grammars, is attractive for semantic analysis due to its transparent syntax-semantic interface. In particular, for a lexical item its syntactic category determines the (semantic) type of its logical form. For instance, the

[2]In this example, a simply typed λ-calculus ($\lambda^{\rightarrow}$), with basic types for entities e and truth values t, is used as a glue language for FOL semantic interpretations. In the calculus, the FOL predicates for lexical entities, e.g. $\mathtt{defeat}(x, y)$, are represented in Curried form, e.g., $\mathtt{defeat}\ x\ y$.

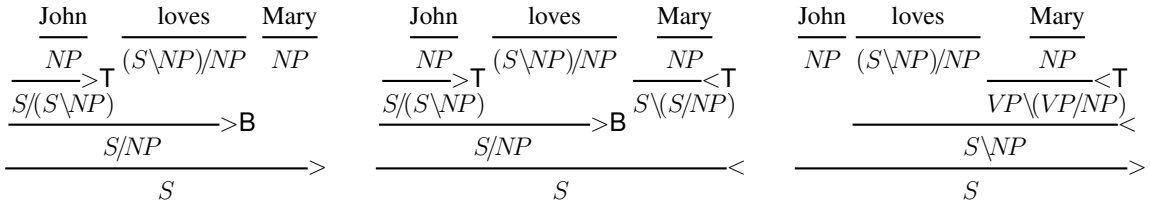


Figure 3.3: Spurious ambiguity in CCG parsing

transitive verbs "*defeats*" and "*overtakes*" both are of category $(S/NP)\backslash NP$ which already predicts the semantic type $e(et)$ of their logical forms. The interface is established by a mapping $\mathcal{T}$ that translates syntactic categories into semantic types as follows:

$$NP \overset{\mathcal{T}}{\mapsto} e, \qquad S \overset{\mathcal{T}}{\mapsto} t, \qquad Y\backslash X \overset{\mathcal{T}}{\mapsto} \mathcal{T}(X)\,\mathcal{T}(Y), \qquad Y/X \overset{\mathcal{T}}{\mapsto} \mathcal{T}(X)\,\mathcal{T}(Y)$$

This transparency between syntactic categories and semantic types in CCG is known as the *principle of categorial type transparency*. In order to project the categorial type transparency for compound phrases, the combinatory rules are subject to a condition called the *principle of combinatory type transparency*. The latter principle forces that semantic interpretations (i.e. logical forms) are combined or modified according to the way their corresponding syntactic categories are combined or modified. For instance, in (>B), logical forms `f` and `g` (or $\lambda P.P$ `B` and $\lambda yx.$ `defeat` $y\,x$ in Figure 3.2) are combined in the same way as their corresponding Z/Y and Y/X syntactic categories (or $S/(S\backslash NP)$ and $(S\backslash NP)/NP$, respectively). Due to these transparency principles, starting only from lexical semantic interpretations, a derivation tree gives a semantic interpretation of the entire phrase for free. The derivation in Figure 3.2 serves as an evidence for this: the final logical form is obtained by faithfully applying the combinatory rules to constituents and carrying out operations of the λ-calculus over logical forms.

From a linguistic perspective, CCG is one of the most well-studied categorial grammar formalism. With functional categories and a few combinatory rules, it is able to analyze a wide range of *bounded* and *unbounded dependencies*—whether a dependent is in the same tensed clause as its head or outside the clause, respectively. The CCG derivation in Figure 3.2 analyzes right node rising which represents an example of an unbounded dependency as the dependent "*Real Madrid*" is outside the verbal clauses. Further analysis of bounded (e.g., reflexivization, dative shift, heavy NP shift, object and subject control, etc.) and unbounded constructions (e.g., extraction in Figure 3.4b, argument cluster coordination, right-node raising in Figure 3.2, etc) can be found in Steedman (1996), Steedman (2000), or in a shorter introduction to CCG by Steedman and Baldridge (2011).[3]

There can be several CCG derivations for a single unambiguous sentence, where all these derivations lead to the same logical form (see Figure 3.3). This kind of ambiguity is called *spurious* and it is essential for CCG due to the type-raising and functional composition rules. For instance, using these combinatory rules (and forbidding double

[3]For the gapping constructions like "*Three points Real Madrid got without deserving*" and "*Barcelona defeated Real Madrid and Atletico Valencia*" CCG needs extra combinatory rules *substitution* and *decomposition*, respectively. While the former rule is rarely used in other constructions, the latter one is considered controversial as it violates certain principles of CCG.

use of the type-raising rule on the same category), there are six derivations for a simple sentence like "*John loves Mary*" with fixed lexical categories. Figure 3.3 lists three non-trivial derivations from those six. In general, the number of derivations over fixed lexical categories increases exponentially with the number of lexical elements.[4]

For more details about the CCG formalism and its linguistic and computational applications, we recommended to consult (Steedman, 1996), (Steedman, 2000), or Steedman and Baldridge (2011). The latter gives a relatively short introduction to all these aspects of CCG. The next section describes the state-of-the-art wide-coverage CCG parsers which will be used in generation of LLFs.

3.2 Wide-coverage CCG parsers

One of the main reasons why the CCG formalism is chosen for obtaining LLFs is that there are at least two robust and accurate wide-coverage parsers for it. In order to make the text self-contained, this section describes those two CCG parsers, the C&C tools (Clark and Curran, 2007) and EasyCCG (Lewis and Steedman, 2014a), provides several qualitative or quantitative characteristics for them and gives examples of actual derivation trees. The information about each parser helps to understand why specific parsing mistakes are systematic and also sheds light on structures of generated LLFs.

3.2.1 The C&C tools

The C&C tools by Clark and Curran (2007) is a collection of NLP systems such that their pipeline, called the C&C parser, can efficiently analyze a wide-coverage text in a CCG-style. The C&C parser (sometimes in short, C&C) by default expects a tokenized input in the style of the Penn Treebank (Marcus et al., 1993)[5] and returns analyses in one of the several formats: CCG predicate-argument dependencies, Briscoe and Carroll style grammatical relations (Carroll et al., 1998), or CCG derivations rendered as a Prolog term. Since the parser is statistical, for every input it returns a derivation with the highest probability. The NLP systems of the C&C parser can be used as a separate tool or as a part of the pipeline. Below we list and briefly describe each of these systems:

- The **POS tagger** (Curran and Clark, 2003a) annotates each token with one of the part of speech (POS) tags used in the Penn Treebank;[6] its accuracy is slightly more than 97%. The tools also contain a multi-POS tagger that, instead of a single most probable POS tag, returns a list of POS tags the probabilities of which are above some predefined threshold.

[4] Despite the spurious ambiguity in CCG parsing, there exist methods that avoid this problem. Existing CCG parsers are designed in such a way that they aim to return a *normal form* derivation—a derivation using the least number of the type-raising and functional composition rules. In order to do so, a parser ignores those sub-derivations that either get low probabilities or logical forms of which are already obtained by another *simpler* sub-derivation.

[5] https://www.cis.upenn.edu/~treebank/tokenization.html

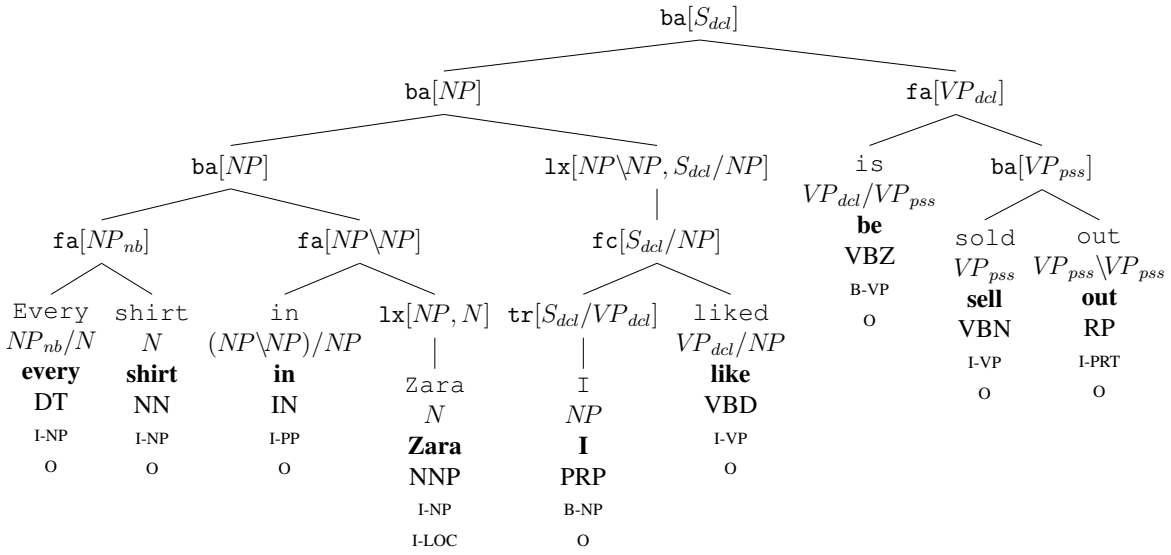
[6] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

- The **chunker** tries to identify spans that have to be constituents. Constraints put by the chunker make parsing search more efficient (Djordjevic et al., 2007).
- The **named entity recognizer** (NER) (Curran and Clark, 2003b) identifies tokens as a part of a proper name; it distinguishes seven types of proper names. Tags for a person (`PER`), organization (`ORG`), location (`LOC`) and date (`DAT`) are among them.
- The **morphological analyser Morpha** (Minnen et al., 2001) is used as a lemmatizer: it annotates each token with a lemma.
- The **supertagger** is a statistical tagger that tags each token with a CCG category, i.e. a *lexical category*. The decision of the supertagger is based on the following information: (i) a token's lexicon of lexical categories (i.e. a set of categories assigned to the token in a training data), (ii) the words and their POS tags in the 5-word window with the token in the center, and (iii) lexical categories of two previous words (Clark and Curran, 2004a). With respect to the Wall Street Journal text, the accuracy of the supertagger (based on the first highest rank lexical category) per word is around 91.5% while the sentence accuracy (i.e. tagging all words in a sentence correctly) is 32.5% (Clark and Curran, 2007, p. 29). The supertagger can also serve as a multi-supertagger—assigning a list of most probable lexical categories per word.
- The **CCG parser** is a statistical parser that attempts to build a CCG derivation tree over lexical categories assigned by the supertagger. Not all combinatory rules used by the parser are found in the CCG formalism or vice versa. For example, the parser uses *type-changing* rules, also called *lexical rules*, unfamiliar for CCG (see `lx` rule in Figure 3.4) which destroys the transparency principles of CCG. Moreover, the type-raising rule ($>\mathsf{T}$), containing a free meta-variable T, is restricted to certain categories in order to maintain parsing efficiency (Clark and Curran, 2007).[7]

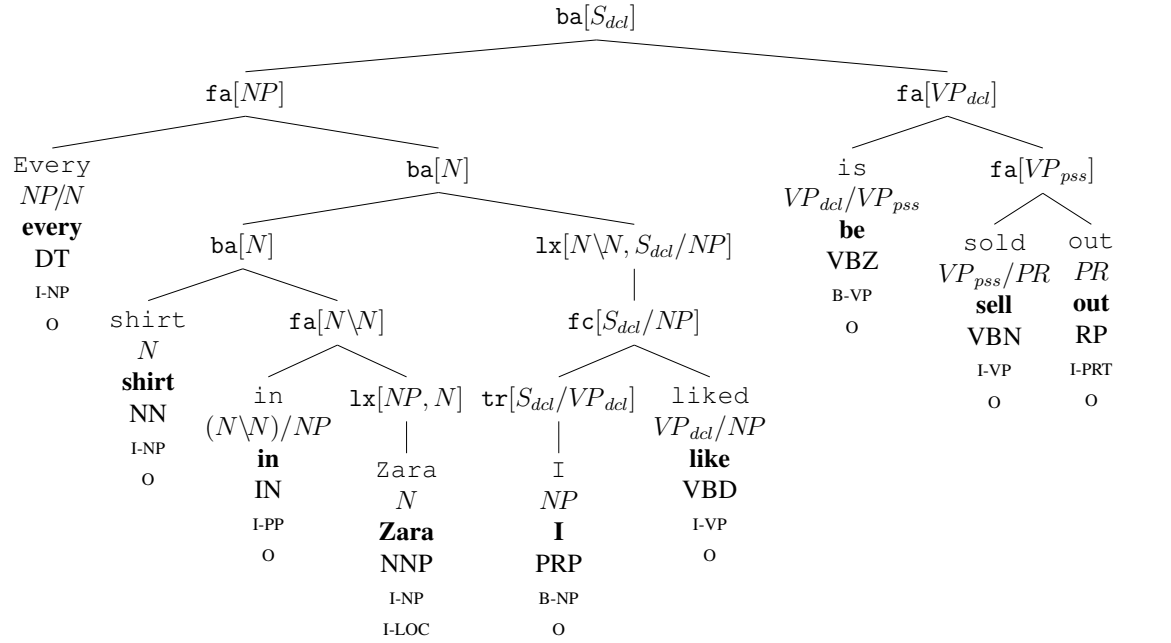
The C&C parser is trained on CCGbank (Hockenmaier and Steedman, 2007)—a treebank of CCG derivations semi-automatically obtained from phrase-structure trees of the Penn Treebank. Therefore, a set of lexical categories and combinatory rules the parser uses are directly coming from CCGbank. The employed lexical categories are built from primitive categories with features. For example, S_{dcl}, S_{ng} and S_{pss} categories are intended for declarative, gerund and passive sentencial phrases, respectively.[8] CCGbank uses 20 combinatory rules with more than 3,000 instances, i.e. combinatory rules with instantiated meta-variables. The statistical model of C&C parser depends on instances of the rules as it is trained on counts of the instances. The C&C parser, trained on Sections 02-21 and developed on Section 00 of CCGbank, can obtain derivations for 99.6% of the

[7]The core components of the C&C parser are the supertagger and the parser which tightly interact with each other during parsing. Since supertagging is considered as *almost parsing*, the supertagger acts as a multitagger and tags tokens with a set of lexical categories. In case the parser cannot find a derivation over a certain set of lexical categories, then it picks less probable categories from the suppertager and tries to find a derivation over them. This interactive approach, called *adaptive supertagging*, results in a higher parsing speed compared to an approach where the parser searches a derivation over entries each assigned a set of all possible lexical categories (Clark and Curran, 2003).

[8]Tne average number of lexical categories with features (i.e. instantiated lexical categories) per word type is greater than 30, where some closed-class words might have much more lexical categories (e.g., "*as*" has 130, "*is*" – 109, "*to*" – 98, etc.).



(a) A CCG tree produced by the C&C parser trained on CCGbank. This model prefers NP-S analysis to Nom-S analysis of a restrictive relative clause and a restrictive noun modification: "*in Zara*" and "*I like*" are of category $NP\backslash NP$. Also a verb particle "*out*" is analyzed as a verb phrase modifier. The category NP_{nb} expresses a non-bare NP.



(b) A CCG tree by rebanked C&C. In contrast to the CCG tree in (a), the tree opts for Nom-S analysis of a restrictive relative clause and a restrictive noun modification. Unlike (a), the current derivation directly yields a correct logical form when combined with standard lexical semantics. Notice also that the verb particle gets a new category PR and serves as a complement to the verb.

Figure 3.4: CCG trees for "*Every shirt in Zara I liked is sold out*" by C&C trained on the original (a) and rebanked (b) CCGbanks. Terminal nodes contain a token, lexical category, lemma, POS tag, and tags from the chunker and the NER; non-terminal nodes are annotated with a combinatory rule (abbreviated according to Table 3.2) and a resulted category. VP_i abbreviates $S_i\backslash NP$ where i stands for a feature.

sentences in the unseen Section 23. But only 33% of the derivations are flawless (Clark and Curran, 2007, p. 36).

There is also a more recent statistical model for the C&C parser which is trained on a rebanked version of CCGbank (Honnibal et al., 2010). The rebanked CCGbank is an updated version of CCGbank with improved analyses of certain constructions.[9] Honnibal et al. (2010) show that the rebanked C&C parser performs only 0.8% worse than the model trained on the original CCGbank, although the new model returns more fine-grained CCG derivations. In order to give an example of a C&C derivation and to contrast the two models of the parser, the CCG derivations of the same sentence based on these models are given in Figure 3.4. We present a CCG derivation produced by CCG parsers as an upside-down tree, called a CCG tree.

The C&C parser was successfully used in several large-scale NLP tasks like textual entailment recognition (Bos and Markert, 2005; Bjerva et al., 2014), semantic similarity (Bjerva et al., 2014; Beltagy et al., 2014), question answering (Clark et al., 2004), or producing semantics of wide-coverage text in combination with Boxer (Bos et al., 2004; Bos, 2008). For more details about the C&C parser, readers are advised to consult (Clark and Curran, 2007), which also contains appendices describing the implementation details.

3.2.2 EasyCCG

EasyCCG, by Lewis and Steedman (2014a), is a CCG parser that has a simpler architecture compared to the C&C parser. It only consists of a simple lemmatizer, a supertagger and a parser. EasyCCG was created under the ideology *supertagging is almost parsing* and its supertagger is the most crucial component of the system.

The supertagger of EasyCCG (Lewis and Steedman, 2014b) is a unigram log-linear tagger that assigns a distribution over all lexical categories to each lexical item. The supertagging model employs features of the words in the ± 3 word context window surrounding a lexical item.[10] The word embeddings used by the supertagger are trained on a large unlabeled corpus; this allows more generalized suppertagging on words not seen in the labeled/annotated data. The main difference from the C&C parser is that the EasyCCG supertagger does not employ information about POS tags. As a result, the supertagger is cheaper to train—it does not require labeled data, generalizes better on the unseen data, and is not influenced by the errors coming from POS tagging.

The parser component is non-statistical: combinatory rules are not distinguished from each other based on their frequency. The probability of a derivation is merely a product of the probabilities of the lexical categories. For building a derivation, the parser uses A* search which potentially allows the parser to consider all categories for each lexical item until it finds a derivation. In this way, the search space is not pruned by the supertagger as it is done in the C&C parser. In case there are several possible derivations over the same sequence of lexical categories, the parser prefers a derivation with the maximum length

[9]The CCG analyses are improved for verb predicate-argument and noun predicate-argument constructions, verb-particle constructions, compound nouns, partitive constructions, punctuations, and restrictive and non-restrictive nominal modifiers.

[10]The word features are 2-character suffixes, capitalization and 50-dimensional word embeddings (Turian et al., 2010), where the latter being a key feature of the supertagger. A word embedding represents a high-dimensional vector extracted from a large corpus. The vector encodes frequencies of the words that occur in the context of a target word.

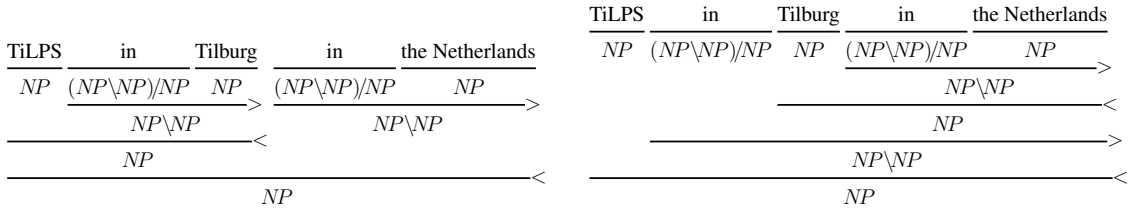


Figure 3.5: Different derivations with different logical forms. The length of dependency arcs in (a) is longer than in (b); therefore EasyCCG prefers (a) to (b) if it derives both.

of arcs in the corresponding dependency tree. Consequently non-local attachments (a) are favored over local ones (b) in the derivations (see Figure 3.5). This heuristic was chosen simply because it performed better on development data.[11]

EasyCCG uses fewer combinatory rules compared to the C&C parser. In total there are 23 instantiated or meta-rules in the grammar, where 13 of them are unary. Similarly to C&C, EasyCCG contains type-changing (i.e. lexical) rules, in total 10. For instance, these rules carry out the following type-changes:

$$N \overset{\texttt{lx}}{\longmapsto} NP, \quad S\backslash NP \overset{\texttt{lx}}{\longmapsto} NP\backslash NP, \quad S_{to}\backslash NP \overset{\texttt{lx}}{\longmapsto} N\backslash N, \quad S_{dcl}/NP \overset{\texttt{lx}}{\longmapsto} NP\backslash NP, \quad S\backslash NP \overset{\texttt{lx}}{\longmapsto} S/S$$

Hence, the transparency principles of CCG are also violated by EasyCCG.

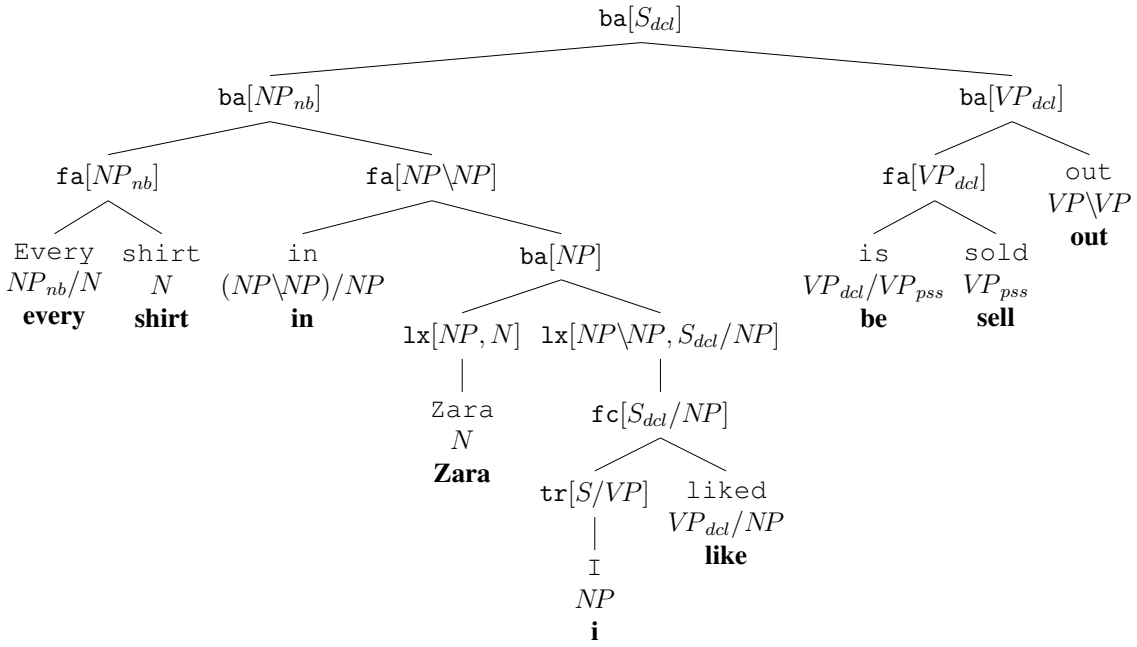
Compared to the C&C parser, EasyCCG is *leaner*, 4 times faster, cheaper to train and generalizes better on unseen data. After training it on CCGbank, EasyCCG achieves slightly better results than C&C.[12] EasyCCG expects as an input a natural language text (not necessarily tokenized) and can return n-best CCG derivations of the input. The two best parse trees by EasyCCG for the sentence discussed in Figure 3.4 are given in Figure 3.6. Although EasyCCG and C&C assign almost identical lexical categories for the sentence, the derivation trees are different. The trees by EasyCCG failed to capture a long distance dependency and gave a different syntactic reading of a sentence. Despite this failure, EasyCCG is still able to capture analyses of Figure 3.4 using the n-best feature. The trees produced by EasyCCG are free from POS and NER tags as it does not require these tags. Nevertheless, EasyCCG allows an input augmented with POS and NER tags (e.g., from the C&C tools) where the tags are later copied in the final tree. In contrast to Morpha (Minnen et al., 2001), used in C&C, EasyCCG uses a less accurate morphological analyzer.

Apart from the n-best derivation feature, EasyCCG also allows to constrain the root category of an input sentence: to be either a declarative sentence, a question, or a noun phrase. Unfortunately, these two features are not available in the C&C parser; this gives EasyCCG certain advantages over C&C from application perspectives.

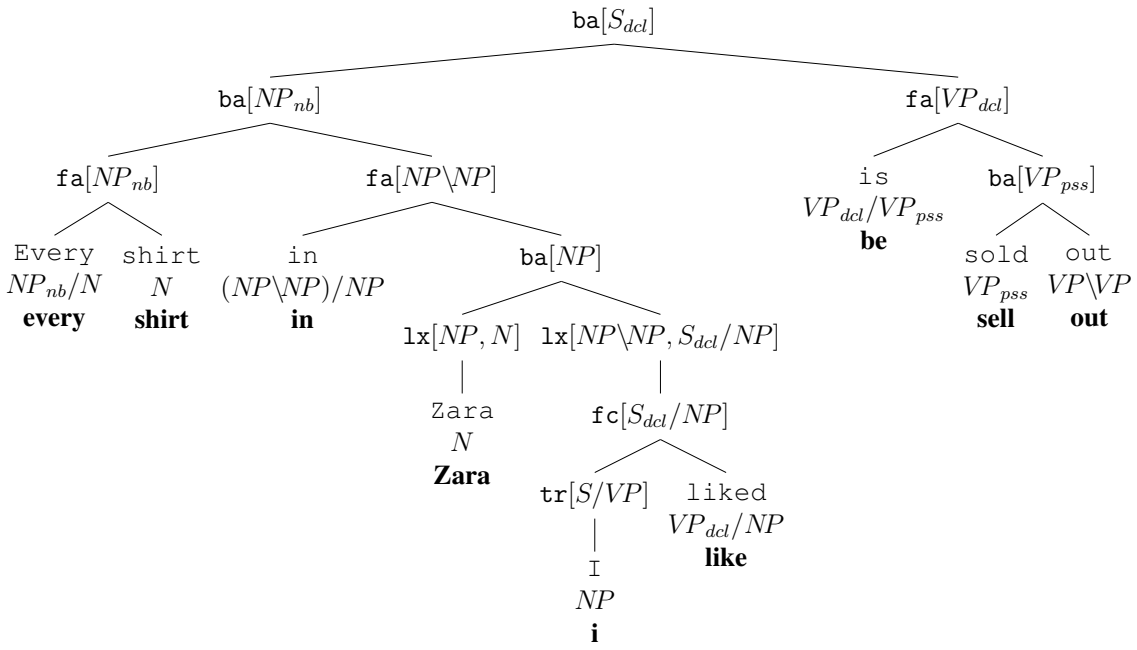
[11] Although there is nothing more said on this issue by Lewis and Steedman (2014b), one of the reasons why favoring non-local attachments shows better results could be that parsing is the only phase where non-local attachments can be encouraged. Notice that lexical categories are primed for local attachments as the supertagger is trained on local ± 3 word context windows.

[12] Lewis and Steedman (2014a) report that the best results are achieved when EasyCCG's supertagger is combined with the parser component of C&C.

[12] These parse trees are obtained using EasyCCG version 0.2 with pre-trained models from http://homepages.inf.ed.ac.uk/s1049478/easyccg.html



(a) The first best derivation tree by EasyCCG, where a reduced relative clause is attached to "*Zara*" and a verb particle "*out*" modifies a compound verb phrase.



(b) The second best derivation tree by EasyCCG. The tree differs from the best one (a) only in the analysis of the final verb phrase.

Figure 3.6: The first (a) and the second (b) best trees for "*Every shirt in Zara I liked is sold out*" by EasyCCG. The trees are significantly different from the trees produced by the C&C parser (Figure 3.4). EasyCCG failed to capture a long-distance dependency of "*I like*" on "*shirt*". It also preferred NP-S analysis to Nom-S as the considered version of EasyCCG is trained on CCGbank. The combinatory rules are abbreviated according to Table 3.2. VP_i abbreviates $S_i\backslash NP$ where i stands for a feature.

All in all, EasyCCG is another robust parser that parses sentences according to the CCG formalism. It is made publicly available for download[13] and for trying online[14].

3.3 From CCG derivations to CCG terms

Two types of directional slashes and a forward-backward dichotomy in combinatory rules represent a technique that controls word order on the surface level. From a semantic perspective, this directional division is redundant. It does not matter for the semantic compositional process whether an argument is found on the left or right of a function; what matters is that a function finds and applies to an argument, which can be expressed with a canonical order—an argument directly following a function. In this section, our goal is to abstract from directional elements of CCG and produce a structure that is more universal and adequate as a semantic representation.

In order to eliminate directional slashes in categories, directional categories will be converted to non-directional ones. In other words, both slashes will be replaced by a single type/category constructor. Obviously, in this process only functional categories will be affected. For instance, a syntactic category $(S\backslash NP)/\!/NP$ for transitive verbs will be changed into a functional category $(NP,(NP,S))$, where the comma is a category constructor. We call these non-directional syntactic categories *syntactic types* and, hereafter, represent them like $(\mathtt{np},(\mathtt{np},\mathtt{s}))$. For a syntactic category, a corresponding syntactic type is obtained by replacing both slashes with the comma and simply reversing an order of primitive categories while maintaining the grouping enforced by parentheses.[15] Reversing is a result of an opposite order in argument-value pairs of CCG categories and syntactic types: value/argument and value\argument in contrast to (argument,value), respectively. The conversion is demonstrated by the following example, where digits are considered as primitive categories:

$$\big((1/2)\backslash(3\backslash4)\big)/(5/6) \longmapsto \big(6,5\big),\big((4,3),(2,1)\big)$$

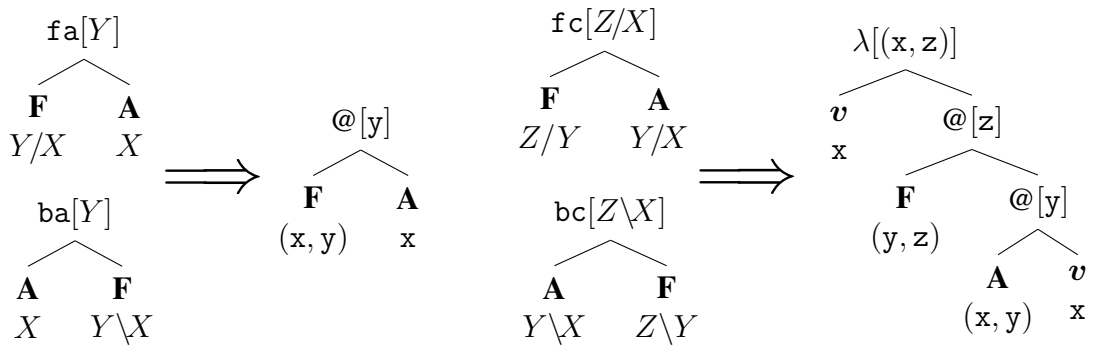
Apart from changing categories, the directionality has to be discarded from combinatory rules as well. In particular, since a functional syntactic type does not contain the information on which side it has to find an argument, a function-argument order over constituents has to be altered in such a way that the application obeys typing. Figure 3.7 shows examples of rearranging nodes for the functional application and functional composition rules. In the resulted structures, the lexical entries with new syntactic types are combined using function application and λ-abstraction of the simply typed λ-calculus.

It is not a coincidence that resulted structures in Figure 3.7 represent simply typed λ-terms. Indeed, our aim is to transform CCG derivations into λ-terms and CCG categories into syntactic types. In order to design a general procedure of transforming a CCG derivation into a λ-term, for each CCG combinatory rule we define how its constituents are shuffled (similarly to the transformations in Figure 3.7). Rearrangement of

[13] https://github.com/mikelewis0/easyccg/releases

[14] http://homepages.inf.ed.ac.uk/s1049478/easyccg.html

[15] Correctness of this transformation can be formally proved using mathematical induction over the length of a category, where the length represents the number of primitive categories.



(a) A structure for the functional application rule fa coincides with the final structure due to the convention associated to a term application @, where a function precedes its argument.

(b) A hypothesis about existence of an item of category X, which the functional composition rules make, is represented by a variable $\boldsymbol{x}$ of category X. After composition of initial constituents, the hypothetical $\boldsymbol{x}$ is taken away via the λ-abstraction.

Figure 3.7: Rearranging nodes for the functional application and functional composition combinatory rules in order to accommodate syntactic types. Non-terminal nodes are labeled with a category or type and an operation (e.g., fa – the forward application rule, @ – the functional application of λ-terms, or λ – the λ-abstraction operation.

constituents follows the way logical forms are combined by the combinatory rules.[16] For example, the obtained structures in Figure 3.7 are presented in the (underlined) semantic interpretations of the combinatory rules:

$$Y/X : \mathtt{f} \quad X : \mathtt{a} \;\Rightarrow\; Y : \underline{\mathtt{fa}} \qquad \textit{Forward functional application} \qquad (>)$$

$$X : \mathtt{a} \quad Y\backslash X : \mathtt{f} \;\Rightarrow\; Y : \underline{\mathtt{fa}} \qquad \textit{Backward functional application} \qquad (<)$$

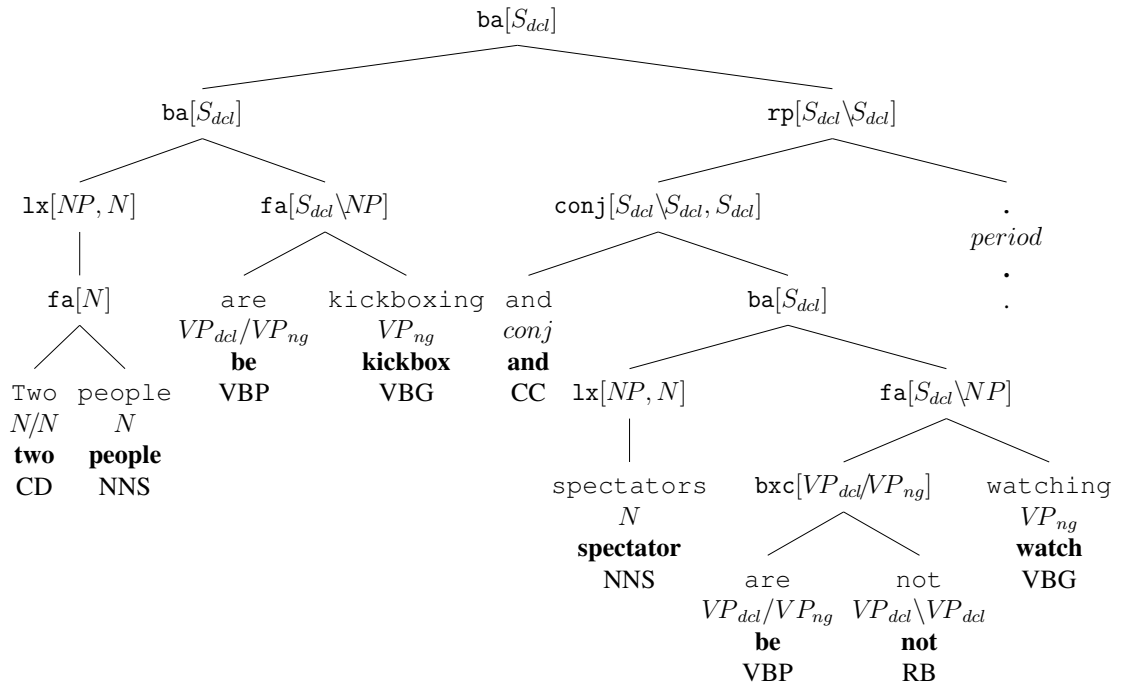
$$Z/Y : \mathtt{f} \quad Y/X : \mathtt{g} \;\Rightarrow\; Z/X : \underline{\lambda x.\, \mathtt{f}(\mathtt{g}x)} \qquad \textit{Forward functional composition} \qquad (>\mathsf{B})$$

$$Y\backslash X : \mathtt{g} \quad Z\backslash Y : \mathtt{f} \;\Rightarrow\; Z\backslash X : \underline{\lambda x.\, \mathtt{f}(\mathtt{g}x)} \qquad \textit{Backward functional composition} \qquad (<\mathsf{B})$$

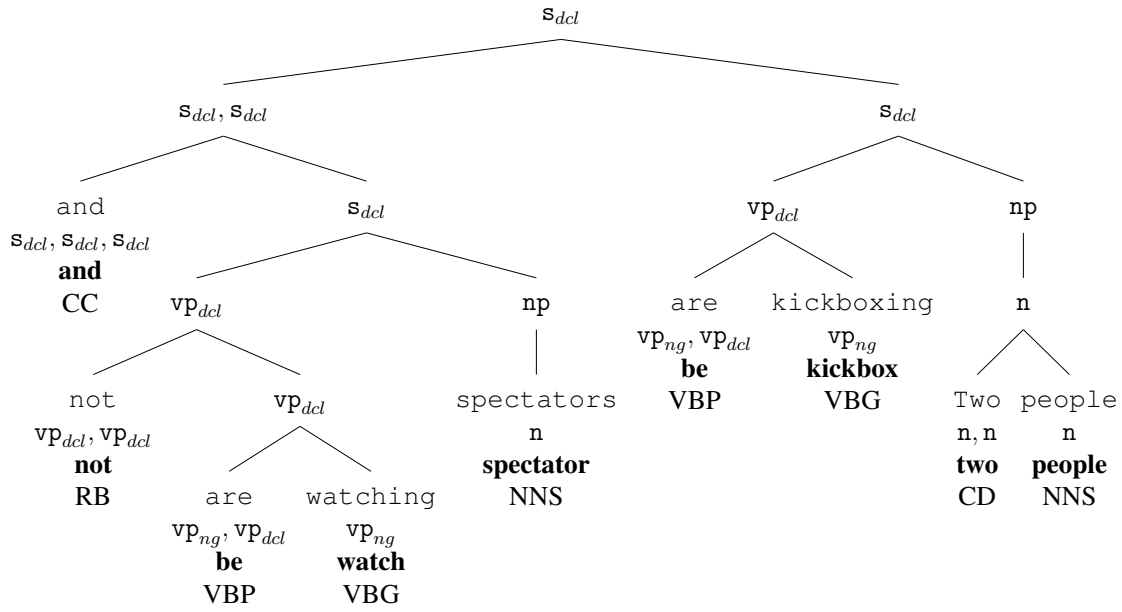
Since the CCG parsers use combinatory rules that are not in the CCG formalism, it is necessary to define transformations for them, too. In Table 3.2, almost all combinatory rules of the CCG parsers with corresponding transformations are given. The transformation schemata in Table 3.2 can be also seen as a rule of combining logical forms: if constituents in a schema are replaced by their semantic counterparts then semantics of a resulted constituent is obtained.

As the parsers employ the lexical rules, it is not straightforward to convert CCG derivations into λ-terms. To make the conversion procedure step-wise and transparent, first, a derivation is translated into a term with type-changes, and then the type-changes are eliminated via a correction procedure. A *CCG term* is a typed λ-term that, in addition to the functional application and the λ-abstraction, uses type-changing operators $[\,.\,]^{\beta}_{\alpha}$ for any α and β types. A type-changing operator $[\,.\,]^{\beta}_{\alpha}$ is simply a constant λ-term of type

[16] Bos (2009) does a related job. On top of the CCG derivations of the C&C parser, he builds semantic interpretations in Discourse Representation Theory (DRT) (Kamp and Reyle, 1993) while using the λ-calculus as a combining device. The semantic interpretations, i.e. the terms of λ-DRT, are combined according to the CCG combinatory rules. In contrast to (Bos, 2009), we do not build any layer for semantics on top of the derivations, but we convert derivations into a new semantic layer.



(a) A CCG derivation by the rebanked C&C parser. Lexical entries are annotated with syntactic categories, lemmata and POS tags. Non-terminal nodes are labeled with a combinatory rule (written in a Prolog style) and a final category that is optionally followed by a source category (e.g., a source category is present in `lx` and `conj`).



(b) A CCG term obtained from the CCG tree in (a) by removing directionality in CCG categories, rearranging lexical entries according to Table 3.2 and carrying out β-reduction.

Figure 3.8: A CCG tree (a) and the corresponding CCG term (b) for a sentence "*Two people are kickboxing and spectators are not watching.*" from the SICK dataset (Marelli et al., 2014b). VP_i and $\texttt{vp}_i$ abbreviate $S_i\backslash NP$ and $(\texttt{np}, \texttt{s}_i)$ respectively.

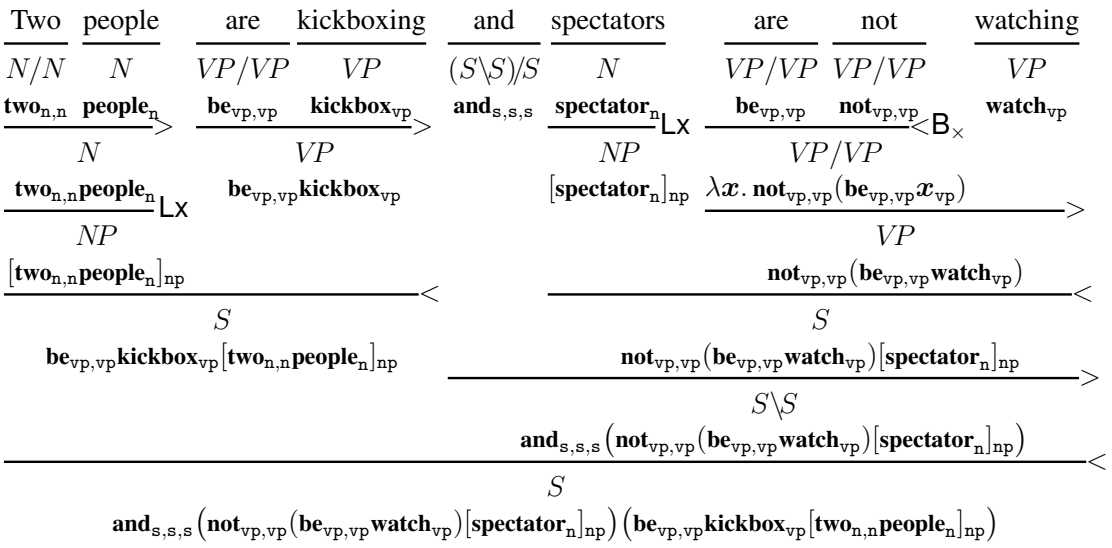


Figure 3.9: A CCG derivation of the sentence in Figure 3.8, where each lexical entry has a canonical logical form—a term expressed by the corresponding lemma typed with the corresponding syntactic type. The logical form of the sentence coincides with the CCG term (2) of Figure 3.8b.

(α, β). Hence, their introduction in the calculus is straightforward. For brevity, we employ a polymorphic notation of type-changing operators: we write $[\mathbf{A}]_\beta$ instead of $[\mathbf{A}_\alpha]^\beta_\alpha$. Examples of CCG terms are given in the last column of Table 3.2.

After transforming each combinatory rule of a CCG derivation according to Table 3.2, a corresponding CCG term is obtained. As a demonstration of this transformation, a CCG derivation tree (a) by the C&C parser and its corresponding CCG term (b) are given in Figure 3.8. In general, in order to obtain a concise CCG term, β-reduction needs to be carried out on a term. For example, after faithfully converting each combinatory rule of a constituent "*are not watching*" in Figure 3.8a, a β-redex (1) is obtained due to the `bxc` combinatory rule. So, for obtaining *nice* CCG terms from CCG derivations, one also needs to automatize the λ-calculus.

$$\big(\lambda x.\,\texttt{not}\,(\texttt{are}\,x)\big)\;\texttt{watching} \xrightarrow{\beta} \texttt{not}\,(\texttt{are watching}) \tag{1}$$

Obtained CCG terms are information-rich terms as their constant terms (i.e. lexical items) maintain all annotations they have in a CCG derivation. When a CCG term is written in-line, a lemma of a lexical item abbreviates the entire item as it is shown for the CCG term of Figure 3.8b below:

$$\mathbf{and}_{\mathrm{s,s,s}}\big(\mathbf{not}_{\mathrm{vp,vp}}\big(\mathbf{be}_{\mathrm{vp,vp}}\mathbf{watch}_{\mathrm{vp}}\big)[\mathbf{spectator}_{\mathrm{n}}]_{\mathrm{np}}\big)\big(\mathbf{be}_{\mathrm{vp,vp}}\mathbf{kickbox}_{\mathrm{vp}}[\mathbf{two}_{\mathrm{n,n}}\mathbf{people}_{\mathrm{n}}]_{\mathrm{np}}\big) \tag{2}$$

Finally, a CCG term can be seen as a canonical logical form of a CCG derivation: if each lexical entry has its lemma as a logical form typed with a corresponding unidirectional category, then a CCG term of a CCG derivation coincides with a logical form of a whole phrase. The former fact is demonstrated in Figure 3.9.

3.4 Correcting CCG terms

A CCG term is a typed λ-term (with possible type-changing rules in it) that is directly obtained from a CCG derivation. In order to obtain LLFs suitable for reasoning, we further process CCG terms as follows. First, certain closed-class lexical terms are replaced by their canonical forms, e.g., **all** $\rightsquigarrow$ **every**. Also several multiword expressions, e.g., "*in front of*", are identified and treated as lexical terms. This step serves as a pre-processor which shrinks term vocabulary. The latter itself decreases the number of tableau rules and make tableau rules leaner. Second, we eliminate most frequent type-changes in CCG terms. The changes are explained either by setting new types for lexical entries or by inserting new lexical entries in CCG terms. Third, we fix wrong analyses the CCG parsers systematically produce. Such wrong analyses mainly concern attachments of relative clauses and prepositional phrases.

During all these three steps, CCG terms are modified by term rewriting rules. The rules are collected manually in the data-driven adaptation described in §6.2.1. The whole correction procedure is facilitated by syntactic types and the absence of word order in CCG terms. The former aspect contributes to fine-grained matching between rewriting rules and CCG terms while the latter aspect makes the rules simpler and keeps their number relatively small. The pseudocode of the correction procedure is given in algorithm 1 (Appendix B). Before presenting the actual rules, we first discuss shortcomings of the CCG derivation trees obtained from the CCG parsers and show how they might lead to poor performance on textual entailment recognition.

3.4.1 Shortcomings of the CCG derivations

For reasoning over sentences, it is essential to get the semantics of sentences right. Getting adequate LLFs from CCG derivations is highly important for the natural tableau system to find a proof. Unfortunately, the parsers are not flawless and they make a lot of mistakes. Remember that for CCGbank, C&C and EasyCCG get only about 85% of dependencies right (Lewis and Steedman, 2014a). Moreover, C&C obtains a gold derivation, i.e. a derivation with no mistakes, for less than 33% of the newswire sentences (§3.2.1).

These results seem quite desperate for producing adequate semantic representations from CCG trees, but the situation is still encouraging as the parsers displayed promising results in several semantic tasks (Bjerva et al., 2014; Beltagy and Erk, 2015; Artzi et al., 2015). Moreover, some textual entailment datasets (Cooper et al., 1996; Marelli et al., 2014b), unlike CCGbank, contain simpler sentences. Hence, we expect less parsing errors from the parsers on those datasets. Finally, although correct semantic representations are essential for the proof system, they are not always necessary for recognizing logical relations. Consider a textual entailment problem in PROB-6. Regardless whether the clause "*I liked*" modifies, "*Zara*" (Figure 3.6) or "*shirt*" (Figure 3.4), the entailment relation is still possible to be captured. It is enough to know that "*sold out*" entails "*out of stock*". Even there can be the case that analyses of constituents do not matter for detecting a logical relation. In SICK-42, it is not necessary to analyze the first conjunct of the sentences in order to classify the sentence pair as contradiction.

Every shirt in Zara I liked is sold out

Every shirt in Zara I liked is out of stock

GOLD: ent; PROB-6

Two people are kickboxing and spectators are not watching

Two people are kickboxing and spectators are watching

GOLD: cont; SICK-42

Despite the above-mentioned, adequate LLFs are still crucial for reasoning and a single mistake in a CCG derivation might lead to a wrong decision. An incomplete list of such kind of mistakes made by C&C and EasyCCG is described next.[17] While presenting a textual entailment problem, its sentences are annotated with errors coming from the parsers. We use brackets for constituency marking and they should not be confused with the type changing operators occurring in CCG terms. Syntactic categories for phrases are written in a subscript.

A CCG derivation tree can often suffer because of a wrong syntactic category assigned by a supertagger. For instance, in the premise of SICK-2911, the C&C supertagger wrongly identifies the verb "*putting*" as ditransitive and assigns $(VP_{ng}/NP)/PP$ to it. In the end, this wrong category contributes to VP_{dcl} as the final category of the sentence. After this mistake, the premise and the conclusion are not comparable, hence impossible to classify the textual entailment as contradiction.

[A woman is putting$_{(VP_{ng}/NP)/PP}$ on makeup]$_{VP_{dcl}}$

[There is no woman putting$_{VP_{ng}/PP}$ on makeup]$_{S_{dcl}}$

GOLD: ent; SICK-2911

There is no girl in [white dancing$^{\mathtt{NN}}_{\mathtt{n}}$]$_N$

A girl in [white$_N$]$_{NP}$ is dancing

GOLD: cont; SICK-219

Even a wrong POS tag can cause failure. In the premise of SICK-219, the C&C POS tagger identifies "*dancing*" as a noun (with the POS tag NN). Based on this error, the C&C supertagger assigns a category N to "*white dancing*" which radically changes the meaning of the sentence and makes it impossible to identify contradiction between the premise and the conclusion. The same problem remains for EasyCCG which does not employ a POS tagger but still its supertagger wrongly tags "*dancing*" with N in the first best derivation.

The type-changing rules also pose a problem as it is not clear what they do from a compositional semantics point of view. For example, in SICK-219, the type of "*white*" is changed from N into NP and the changing operation needs to have corresponding semantics.

[17] Described errors are triggered by using rebanked C&C (Honnibal et al., 2010) and EasyCCG version 0.2 trained on CCGbank. For different versions of the parsers or models, the errors might be different.

One of the most frequent mistakes both parsers make is a wrong prepositional phrase (PP) attachment. This problem occurs naturally from ambiguity related to PP attachments. For instance, in the premise of SICK-2772, identifying "*for cooking into a pot*" as a PP constituent and attaching it to the verb phrase "*pouring oil*" makes it more difficult to capture entailment relation. Another example where both C&C and EasyCCG make a wrong PP attachment is SICK-3657: "*with water*" is attached to the noun instead of the verb in the premise. Afterwards, the attachment makes it impossible to recognize the entailment relation.

A person is $\big[$[pouring oil]$_{VP_{ng}/PP}$[for cooking into a pot]$_{PP}\big]$

A person is $\big[$[pouring cooking oil]$_{VP_{ng}/PP}$[into a pot]$_{PP}\big]$

GOLD: ent; SICK-2772

A boy is filling [a pitcher with water]$_{NP}$

A pitcher is being [filled with water]$_{VP_{pss}}$ by a boy

GOLD: ent; SICK-3657

It is well-known that multiword expressions (MWEs) represent a problem for lexicalized parsers (Sag et al., 2001), and this is also the case for the CCG parsers. The parsers often fail to render MWEs as constituents. For instance, expressions like "*because of*", "*according to*", "*next to*", "*in front of*", etc. are not analyzed as constituents. This makes it difficult to capture entailment in the problems like SICK-6096 as the knowledge of "*beside*" and "*next to*" being synonymous does not suffice.

A man is standing beside a birdcage which is large and colorful

The man is standing next [to a bird cage]$_{PP}$

GOLD: ent; SICK-6096

$\big[$[Most$_{N/N}$ Europeans$_N$]$_N\big]_{NP}$ are resident in Europe
[All Europeans]$_{NP}$ are people
[All people who are resident in Europe]$_{NP}$ can travel freely within Europe

$\big[$[Most$_{N/N}$ Europeans$_N$]$_N\big]_{NP}$ can travel freely within Europe

GOLD: ent; FraCaS-26

Finally, the CCG parsers do not analyze quantified noun phrases (NPs) as type-raised NPs, i.e. as phrases of category $S\backslash(S/NP)$ or $S/(S\backslash NP)$. The reason for it is to maintain the parsing process efficient. On the other hand, generalized quantifiers (Montague, 1973; Barwise and Cooper, 1981) nicely capture monotonic features of quantifiers that are important for monotonicity reasoning (see § 1.3). For instance, with the help of monotonic features of "*most*", it is possible to prove the entailment relation in FraCaS-26. Moreover, monotonicity features can often lead to shorter and more *natural* proofs.

All wrong analyses by the CCG parsers are automatically carried from CCG derivations to CCG terms. In order to guarantee more adequate logical forms, we design procedures to fix some of the shortcomings of CCG terms.[18] Before correcting a wrong analysis, first, it has to be found in a term. We find an error by designing a pattern that reliably matches an error and its context. This pattern may include any information about a lexical term (e.g., its POS tag[19] or lemma) or about a compound term (e.g., its syntactic type or structure) that is available in a CCG term.

The correction procedure consists of term rewriting rules. The rules have a form of *antecedent* $\rightsquigarrow$ *consequent*. Meta-variables ranging over lexical terms are written as lowercase letters while meta-variables denoted by upper-case letters range over a set of all CCG terms. In order to have simpler rules, we assume that all properties of lexical terms in an antecedent are preserved in the consequent, unless stated differently. For a demonstration purpose, consider a rule in (3) that type-raises quantified NPs:

$$\boldsymbol{V}_{\mathrm{np,s}}\ (\boldsymbol{q}^{\mathtt{DT}}_{\mathrm{n,np}}\ \boldsymbol{N}_{\mathrm{n}}) \rightsquigarrow \boldsymbol{q}_{\mathrm{n,np,s}}\ \boldsymbol{N}\ \boldsymbol{V} \tag{3}$$

$$\mathbf{walk}_{\mathrm{np,s}}\ \left(\mathbf{every}^{\mathtt{DT}}_{\mathrm{n,np}}\ (\mathbf{tall}_{\mathrm{n,n}}\mathbf{man}_{\mathrm{n}})\right) \rightsquigarrow \mathbf{every}^{\mathtt{DT}}_{\mathrm{n,np,s}}\ (\mathbf{tall}_{\mathrm{n,n}}\ \mathbf{man}_{\mathrm{n}})\ \mathbf{walk}_{\mathrm{np,s}} \tag{3a}$$

The rule reorders terms and changes the type of $\boldsymbol{q}$ while leaving $\boldsymbol{V}$ and $\boldsymbol{N}$ terms unchanged and implicitly assuming the POS tag `DT` for $\boldsymbol{q}$ in the result. New features of terms are explicitly mentioned in the consequent of a rewriting rule. The transformation in (3a) demonstrates the rule (3) in action.

3.4.2 Simplifying CCG terms

The simplification process of CCG terms consists of several rewriting rules. These rules detect certain MWEs and closed class words and replace them with the corresponding simpler and canonical versions. Also false relational nouns are treated as ordinary nouns. The process makes CCG terms simpler, semantically more transparent and adequate for reasoning.

The rules for MWEs identify and convert MWEs into a constant term, i.e. with no internal structure. These rules are hand-crafted for the MWEs occurring frequently in training RTE datasets (see § 6.2.1). For instance, (4–7) represent some of the rules for MWEs. The rules in (4–6) treat specific MWEs frequently found in the SICK dataset (Marelli et al., 2014b). On the other hand, (7) represents more general rule: given a list of $\langle \boldsymbol{b}, \boldsymbol{o} \rangle$ pairs, it identifies several MWEs like "*because of*", "*instead of*" and "*prior to*" (7a) as a constant.

$$\mathbf{in}_{\mathrm{np},\alpha}\ [\mathbf{front}\ (\mathbf{of}\ \boldsymbol{T}_{\mathrm{np}})]_{\mathrm{np}} \rightsquigarrow \mathbf{in\_front\_of}_{\mathrm{np},\alpha}\ \boldsymbol{T} \tag{4}$$

$$\mathbf{next}_{\beta,\gamma}\ (\mathbf{to}\ \boldsymbol{T}_{\alpha}) \rightsquigarrow \mathbf{next\_to}_{\alpha,\gamma}\ \boldsymbol{T} \tag{5}$$

$$\mathbf{a}_{\mathrm{n,np}}\ (\mathbf{few}_{\mathrm{n,n}}\ \boldsymbol{N}_{\mathrm{n}}) \rightsquigarrow \mathbf{a\_few}_{\mathrm{n,np}}\ \boldsymbol{N} \tag{6}$$

[18] Identifying and correcting errors in CCG terms are simpler and more deterministic than doing this in CCG trees. The reason is simple. A CCG term represents a version of a CCG tree that is abstracted from a word order. Therefore, during finding and fixing errors, one does not need to care about a function-argument order or even about types of combinatory rule.

[19] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

$$\boldsymbol{b}_{\mathtt{pp,vp,vp}}\ (\boldsymbol{o}^{\mathtt{IN|TO}}_{\alpha,\mathtt{pp}}\ \boldsymbol{T}_{\alpha}) \rightsquigarrow \boldsymbol{b\_o}_{\alpha,\mathtt{vp,vp}}\ \boldsymbol{T} \tag{7}$$

$$\mathbf{prior}_{\mathtt{pp,vp,vp}}(\mathbf{to}^{\mathtt{TO}}_{\mathtt{np,pp}}(\mathbf{the}_{\mathtt{n,np}}\mathbf{show}_{\mathtt{n}})) \rightsquigarrow \mathbf{prior\_to}_{\mathtt{np,vp,vp}}(\mathbf{the\ show}) \tag{7a}$$

One way of simplifying a CCG term is to decrease the size of vocabulary. For instance, there are several closed-class words, like "*every*" and "*all*", that are similar from semantic perspectives. Substituting these lexical items with a canonical one makes the vocabulary leaner which itself implies less or simpler tableau rules. Such reductions (8a–d) are carried out by the rule in (8) which replaces a lexical term by the similar term of the same type.

$$\boldsymbol{c}_{\alpha} \rightsquigarrow \boldsymbol{d}_{\alpha} \qquad \text{where} \quad \mathtt{similar}(\boldsymbol{c}, \boldsymbol{d}) \tag{8}$$

$$\mathbf{all}_{\mathtt{n,np}} \rightsquigarrow \mathbf{every}_{\mathtt{n,np}} \tag{8a}$$

$$\mathbf{an}_{\mathtt{n,np}} \rightsquigarrow \mathbf{a}_{\mathtt{n,np}} \tag{8b}$$

$$\mathbf{that}_{\mathtt{vp,n,n}} \rightsquigarrow \mathbf{which}_{\mathtt{vp,n,n}} \tag{8c}$$

$$\mathbf{who}_{\mathtt{vp,n,n}} \rightsquigarrow \mathbf{which}_{\mathtt{vp,n,n}} \tag{8d}$$

In the end, a simplified CCG term has more transparent semantics and consists of fewer lexical terms compared to the initial CCG term. Moreover, it is easier to further correct wrong analyses in the simplified terms or to reason over them.

3.4.3 Explaining the type-changing rules

Both wide-coverage CCG parsers, C&C and EasyCCG, employ type-changing rules which have no analogy in the CCG formalism. The rule is adopted by the parsers in order to make the parsing process efficient and robust.[20] The idea behind a type-changing rule is simple—it *mysteriously* changes a syntactic category of a constituent into the category that *better fits* the context. For instance, if a constituent of category N is followed by an intransitive verb phrase of category $S\backslash NP$, then a type-changing rule can change N into NP and allow the backward application rule to proceed further. For CCG derivations and CCG terms, all lexical rules can be expressed as a single schematic rule:

$$X : \mathtt{a} \ \Rightarrow\ Y : \mathtt{b} \qquad \textit{Lexical rule in the CCG style}$$

$$\boldsymbol{A}_{\mathtt{x}} \overset{[\,.\,]_{\mathtt{y}}}{\longmapsto} \boldsymbol{B}_{\mathtt{y}} \qquad \textit{Lexical rule for CCG terms}$$

But, of course, X and Y meta-variables are not ranging over all categories. There is a fixed number of instances for $\langle X, Y\rangle$ pair which induce concrete type-changing rules.

Our goal in this subsection is to define a new logical form $\mathtt{b}$ (or a new CCG term $\boldsymbol{B}$) given all other components of the lexical rule. From the perspective of CCG terms, the goal boils down to define the $[\,.\,]_{\alpha}$ operator in such a way that $\boldsymbol{B}$ is a well-formed λ-term of type y that is free from the type-changing operator. Below we give several rewriting rules that explain type-changes. While doing so, CCG terms are used as operands.

[20] Initially, type-changing rules were introduced in CCGbank in order to facilitate semi-automatic translation of phrase structure trees into CCG derivations.

3.4.3.1 The lexical rule $\mathtt{n} \mapsto \mathtt{np}$

A rule that alters a category N to NP is the most popular instance of the lexical rule in the CCG parsers. Usually this category change occurs for bare NPs (i.e. an NP without an initial determiner or quantifier). Solutions to explain the $\mathtt{n} \mapsto \mathtt{np}$ type-change in CCG terms can be categorized in five classes.

The first class contains a rewriting rule (9) which directly sets np as a syntactic type of lexical NP term while ignoring the previous n type. A set of POS tags, written as a *regular expression*, puts constraints on the antecedent term. The rule covers lexical terms which were identified by the C&C POS tagger as a proper noun or a personal pronoun. The concrete transformations carried out by (9) are given in (9a–9d). Note that the information attached to lexical terms is preserved unless it is replaced by different information. For example, a POS tag of the lexical term $\boldsymbol{c}$ remains unchanged in (9) while the syntactic type changes.

$$[\boldsymbol{c}^{\mathtt{NNP}|\mathtt{NNPS}|\mathtt{PRP}}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \boldsymbol{c}_{\mathtt{np}} \tag{9}$$

$$[\mathbf{Oracle}^{\mathtt{NNP}}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{Oracle}_{\mathtt{np}} \tag{9a}$$

$$[\mathbf{States}^{\mathtt{NNPS}}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{States}_{\mathtt{np}} \tag{9b}$$

$$[\mathbf{he}^{\mathtt{PRP}}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{he}_{\mathtt{np}} \tag{9d}$$

The rule in (10) also acts on lexical terms but, unlike (9), it splits them. The rule is applicable to terms with a determiner POS tag that are possible to be split into a determiner and a noun (10c). The splitting requirement is checked with the help of $\mathtt{split}/3$ predicate which holds for a predefined set of triples. Currently (10) is the only rule in the second class.

$$[\boldsymbol{c}^{\mathtt{DT}}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \boldsymbol{d}^{\mathtt{DT}}_{\mathtt{n,np}}\ \boldsymbol{n}^{\mathtt{NN}}_{\mathtt{np}} \quad \text{where } \mathtt{split}(\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{n}) \tag{10}$$

$$[\mathbf{nobody}^{\mathtt{DT}}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{no}^{\mathtt{DT}}_{\mathtt{n,np}}\ \mathbf{person}^{\mathtt{NN}}_{\mathtt{n}} \tag{10c}$$

The third class consists of the rules that explain the $\mathtt{n} \mapsto \mathtt{np}$ type-change for non-bare compound CCG terms by assigning proper types to the constituent terms. In particular, the rules in (11) and (12) identify noun modifiers as determiners of type $(\mathtt{n}, \mathtt{np})$ that consequently explain the type-change. The examples of CCG terms processed by these rules are given below:

$$[\boldsymbol{c}^{\mathtt{DT}|\mathtt{CD}}_{\mathtt{n,n}}\ \boldsymbol{N}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \boldsymbol{c}_{\mathtt{n,np}}\ \boldsymbol{N} \tag{11}$$

$$[\mathbf{many}^{\mathtt{DT}}_{\mathtt{n,n}}\ \mathbf{red\_apple}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{many}_{\mathtt{n,np}}\ \mathbf{red\_apple}_{\mathtt{n}} \tag{11a}$$

$$[\mathbf{three}^{\mathtt{CD}}_{\mathtt{n,n}}\ \mathbf{red\_apple}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{three}_{\mathtt{n,np}}\ \mathbf{red\_apple}_{\mathtt{n}} \tag{11b}$$

$$[\boldsymbol{e}^{\mathtt{RB}}_{(\mathtt{n,n}),\mathtt{n,n}}\ \boldsymbol{c}^{\mathtt{CD}}_{\mathtt{n,n}}\ \boldsymbol{N}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \boldsymbol{e}_{(\mathtt{n,np}),\mathtt{n,np}}\ \boldsymbol{c}_{\mathtt{n,np}}\ \boldsymbol{N}_{\mathtt{n}} \tag{12}$$

$$[\mathbf{exactly}^{\mathtt{RB}}_{(\mathtt{n,n}),\mathtt{n,n}}\ \mathbf{two}^{\mathtt{CD}}_{\mathtt{n,n}}\ \mathbf{red\_apple}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{exactly}_{(\mathtt{n,np}),\mathtt{n,np}}\ \mathbf{two}_{\mathtt{n,np}}\ \mathbf{red\_apple}_{\mathtt{n}} \tag{12a}$$

The rules from the fourth class pushes the type-change in sub-terms. The first rule (13) pushes the type-change from a pair of a noun and a relative clause to the noun. This

rule is triggered to process a CCG term (13a) for a noun phrase like in "*there is [nobody eating]$_{NP}$*". The example shows how the relative clause gets out of the scope of $[\,.\,]_{\mathtt{np}}$ and leaves explanation of the shifted type-change of $[\mathbf{nobody}_{\mathtt{n}}]_{\mathtt{np}}$ to the rule in (10).

$$\left[[\boldsymbol{V}_{\mathtt{vp}}]_{\mathtt{n},\mathtt{n}}\ \boldsymbol{T}_{\mathtt{n}}\right]_{\mathtt{np}} \rightsquigarrow [\boldsymbol{V}_{\mathtt{vp}}]_{\mathtt{np},\mathtt{np}}\ [\boldsymbol{T}_{\mathtt{n}}]_{\mathtt{np}} \tag{13}$$

$$\left[[\mathbf{eat}_{\mathtt{vp}}]_{\mathtt{n},\mathtt{n}}\ \mathbf{nobody}_{\mathtt{n}}\right]_{\mathtt{np}} \rightsquigarrow [\mathbf{eat}_{\mathtt{vp}}]_{\mathtt{np},\mathtt{np}}\ [\mathbf{nobody}_{\mathtt{n}}]_{\mathtt{np}} \tag{13a}$$

The second rule (14) from this class propagates the lexical rule down to sub-terms of coordination as it is shown by (14a). Explanation of the type-changes in the sub-terms is later done by the above described rules. In case of (14a), the both sub-terms could be further processed by (9).

$$[\boldsymbol{c}_{\mathtt{n},\mathtt{n},\mathtt{n}}\ \boldsymbol{X}\ \boldsymbol{Y}]_{\mathtt{np}} \rightsquigarrow \boldsymbol{c}_{\mathtt{np},\mathtt{np},\mathtt{np}}\ [\boldsymbol{X}]_{\mathtt{np}}\ [\boldsymbol{Y}]_{\mathtt{np}} \tag{14}$$

$$[\mathbf{and}_{\mathtt{n},\mathtt{n},\mathtt{n}}\ \mathbf{John}_{\mathtt{n}}\ \mathbf{Mary}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{and}_{\mathtt{np},\mathtt{np},\mathtt{np}}\ [\mathbf{John}]_{\mathtt{np}}\ [\mathbf{Mary}]_{\mathtt{np}} \tag{14a}$$

Rules of the last class explain the type-change by inserting a determiner of type $(\mathtt{n},\mathtt{np})$. The inserting procedure is intended for bare compound CCG terms and takes into account a POS tag of a head word of a CCG term. The following simple heuristic is used for detecting a head of a phrase in a CCG tree: (i) a head of a lexical term is itself the same term; (ii) a head of a CCG term does not change during λ-abstraction and type-changing; (iii) for the term application, the head of a resulted term coincides with the head of a function term unless the function is of type (α,α). In that case the head of an argument becomes the head of the resulted term.

If the head of a noun term is singular, marked with `NN`, then a term $\mathbf{a}_{\mathtt{n},\mathtt{np}}$ standing for a canonical indefinite determiner is inserted (15); otherwise a term $\mathbf{s}_{\mathtt{n},\mathtt{np}}$ is used that stands for a plural morpheme (16). In case of inserting the plural morpheme, the POS tag of the head is set to `NN`, a singular noun.

$$[\boldsymbol{T}_{\mathtt{n}}^{\mathtt{HeadPOS:NN}}]_{\mathtt{np}} \rightsquigarrow \mathbf{a}_{\mathtt{n},\mathtt{np}}\ \boldsymbol{T} \tag{15}$$

$$[\mathbf{ice}_{\mathtt{n}}^{\mathtt{NN}}]_{\mathtt{np}} \rightsquigarrow \mathbf{a}_{\mathtt{n},\mathtt{np}}\ \mathbf{ice}_{\mathtt{n}} \tag{15a}$$

$$[\boldsymbol{T}_{\mathtt{n}}^{\mathtt{HeadPOS:NNS}}]_{\mathtt{np}} \rightsquigarrow \mathbf{s}_{\mathtt{n},\mathtt{np}}\ \boldsymbol{T}^{\mathtt{HeadPOS:NN}} \tag{16}$$

$$[\mathbf{red}_{\mathtt{n},\mathtt{n}}\ \mathbf{car}_{\mathtt{n}}^{\mathtt{NNS}}]_{\mathtt{np}} \rightsquigarrow \mathbf{s}_{\mathtt{n},\mathtt{np}}\ (\mathbf{red}_{\mathtt{n},\mathtt{n}}\ \mathbf{car}_{\mathtt{n}}^{\mathtt{NN}}) \tag{16a}$$

The idea behind inserting $\mathbf{a}_{\mathtt{n},\mathtt{np}}$ is that it plays a role of a higher-order existential quantifier. So, despite a term expressing a countable or a non-countable noun, in a positive context with $\mathbb{T}$, existence of an entity in the extension of the noun will be asserted. A plural determiner $\mathbf{s}_{\mathtt{n},\mathtt{np}}$ has the similar function with addition that potentially it can introduce more than one such kind of item. The inserted determiners play a role of a type-shifter in the sense of Partee (1987). The insertion rules (15) and (16) can be considered as the last aid to explain $\mathtt{n} \mapsto \mathtt{np}$ change. Due to this nature, they must have the lowest priority than any other rule for fixing type-changes (see algorithm 1 in Appendix B); otherwise these rules can lead to insertion of redundant determiners.

In order to eliminate the $\mathtt{n} \mapsto \mathtt{np}$ type-change in a CCG term, we apply (9–16) rules to it until they are applicable. We do so by starting it from the root of a term down to its leaves. Notice that the rules does not guarantee that the processed CCG term will be free

Clause	Noun phrase	CCG term
Adjectival (adj)	a glass full of water	$\mathbf{a}_{\mathtt{n},\mathtt{np}}\ \left([\mathbf{full\_of\_water}_{\mathtt{np},\mathtt{s}_{adj}}]_{\mathtt{n},\mathtt{n}}\ \mathbf{glass}_{\mathtt{n}}\right)$
Declarative (dcl)	a glass John broke	$\mathbf{a}_{\mathtt{n},\mathtt{np}}\ \left([\mathbf{John\_break}_{\mathtt{np},\mathtt{s}_{dcl}}]_{\mathtt{n},\mathtt{n}}\ \mathbf{glass}_{\mathtt{n}}\right)$
Passive (pss)	a glass broken by John	$\mathbf{a}_{\mathtt{n},\mathtt{np}}\ \left([\mathbf{break\_by\_John}_{\mathtt{np},\mathtt{s}_{pss}}]_{\mathtt{n},\mathtt{n}}\ \mathbf{glass}_{\mathtt{n}}\right)$
Participle (ng)	a glass falling on the floor	$\mathbf{a}_{\mathtt{n},\mathtt{np}}\ \left([\mathbf{fall\_on\_the\_floor}_{\mathtt{np},\mathtt{s}_{ng}}]_{\mathtt{n},\mathtt{n}}\ \mathbf{glass}_{\mathtt{n}}\right)$

Table 3.1: Examples for the lexical rule changing $(\mathtt{np},\mathtt{s})$ into $(\mathtt{n},\mathtt{n})$

from $\mathtt{n} \mapsto \mathtt{np}$. For instance, in a few case, the C&C parser wrongly identifies the indefinite determiner as of category N and later changes the category into NP. Obviously, in such cases the introduced rules are not able to eliminate the type-change.

3.4.3.2 The lexical rules $\mathtt{vp} \mapsto \mathtt{n}$ and $\mathtt{vp} \mapsto (\mathtt{np},\mathtt{np})$

The second and third most frequent instances of the type-changing rule change $S\backslash NP$ (abbreviated as VP) into $N\backslash N$ or $NP\backslash NP$. Choice between $N\backslash N$ and $NP\backslash NP$ depends on whether an adnominal clause is a restrictive or a non-restrictive modifier of an NP, respectively.[21] A reduced clause that undergoes these type-changes might get the following sentential category features *dcl, ng, pss*, and *adj*; see the examples in Table 3.1.

In order to explain the $\mathtt{vp} \mapsto (\mathtt{n},\mathtt{n})$ type-change in CCG terms, a canonical term **which** of type $(\mathtt{vp},\mathtt{n},\mathtt{n})$, standing for a relative pronoun, is inserted. This is done by the rule in (17) which operates on any intransitive VP (e.g., of type $\mathtt{vp}_{adj}, \mathtt{vp}_{dcl}, \mathtt{vp}_{pss}, \mathtt{vp}_{ng}$):

$$[\boldsymbol{V}_{\mathtt{vp}}]_{\mathtt{n},\mathtt{n}}\ \boldsymbol{N}_{\mathtt{n}} \rightsquigarrow \mathbf{which}_{\mathtt{vp},\mathtt{n},\mathtt{n}}\ \boldsymbol{V}_{\mathtt{vp}}\ \boldsymbol{N} \tag{17}$$

$$[\mathbf{John\_break}_{\mathtt{vp}}]_{\mathtt{n},\mathtt{n}}\ \mathbf{glass}_{\mathtt{n}} \rightsquigarrow \mathbf{which}_{\mathtt{vp},\mathtt{n},\mathtt{n}}\ \mathbf{John\_break}_{\mathtt{vp}}\ \mathbf{glass}_{\mathtt{n}} \tag{17b}$$

The type-change $\mathtt{vp} \mapsto (\mathtt{np},\mathtt{np})$ is treated in the similar way by (18) where the inserted relative pronoun is of type $(\mathtt{vp},\mathtt{np},\mathtt{np})$. Note that this rule can also apply to the result of (13).

$$[\boldsymbol{V}_{\mathtt{vp}}]_{\mathtt{np},\mathtt{np}}\ \boldsymbol{T}_{\mathtt{np}} \rightsquigarrow \mathbf{which}_{\mathtt{vp},\mathtt{np},\mathtt{np}}\ \boldsymbol{V}_{\mathtt{vp}}\ \boldsymbol{T}_{\mathtt{np}} \tag{18}$$

$$[\mathbf{run}_{\mathtt{vp}}]_{\mathtt{np},\mathtt{np}}\ (\mathbf{every}_{\mathtt{n},\mathtt{np}}\ \mathbf{girl}_{\mathtt{n}}) \rightsquigarrow \mathbf{which}_{\mathtt{vp},\mathtt{np},\mathtt{np}}\ \mathbf{run}_{\mathtt{vp}}\ (\mathbf{every}_{\mathtt{n},\mathtt{np}}\ \mathbf{girl}_{\mathtt{n}}) \tag{18b}$$

In order to show how (9–18) rules work hand-in-hand, a CCG tree (19) of a noun phrase *"somebody falling on ice"* is processed below, where the final CCG term (19d) is free of type-changing rules.

$$\left[\left[\mathbf{on}^{\mathtt{IN}}_{\mathtt{np},\mathtt{vp}_{ng},\mathtt{vp}_{ng}}\ \left[\mathbf{ice}^{\mathtt{NN}}_{\mathtt{n}}\right]_{\mathtt{np}}\ \mathbf{fall}^{\mathtt{VBG}}_{\mathtt{vp}_{ng}}\right]_{\mathtt{n},\mathtt{n}}\ \mathbf{somebody}^{\mathtt{DT}}_{\mathtt{n}}\right]_{\mathtt{np}} \tag{19}$$

applying (13) to the whole term $\Downarrow$

$$\left[\mathbf{on}^{\mathtt{IN}}_{\mathtt{np},\mathtt{vp}_{ng},\mathtt{vp}_{ng}}\ \left[\mathbf{ice}^{\mathtt{NN}}_{\mathtt{n}}\right]_{\mathtt{np}}\ \mathbf{fall}^{\mathtt{VBG}}_{\mathtt{vp}_{ng}}\right]_{\mathtt{np},\mathtt{np}}\ \left[\mathbf{somebody}^{\mathtt{DT}}_{\mathtt{n}}\right]_{\mathtt{np}} \tag{19a}$$

[21] Apart from this distinction, there is also a language factor when it comes to non-restrictive adnominals: while the Nom-S analysis, with $N\backslash N$, is argued to be suitable for some languages, the NP-S analysis, with $NP\backslash NP$, is suitable for other languages (Bach and Cooper, 1978).

applying (18) to $[\textbf{on ice fall}]_{\texttt{np,np}}$ $\Downarrow$

$$\textbf{which}^{\texttt{WDT}}_{\texttt{vp,np,np}}\ \left(\textbf{on}^{\texttt{IN}}_{\texttt{np,vp}_{ng},\texttt{vp}_{ng}}\ [\textbf{ice}^{\texttt{NN}}_{\texttt{n}}]_{\texttt{np}}\ \textbf{fall}^{\texttt{VBG}}_{\texttt{vp}_{ng}}\right)\ [\textbf{somebody}^{\texttt{DT}}_{\texttt{n}}]_{\texttt{np}} \tag{19b}$$

applying (15) to $[\textbf{ice}]_{\texttt{np}}$ $\Downarrow$

$$\textbf{which}^{\texttt{WDT}}_{\texttt{vp,np,np}}\ \left(\textbf{on}^{\texttt{IN}}_{\texttt{np,vp}_{ng},\texttt{vp}_{ng}}\ (\textbf{a}^{\texttt{DT}}_{\texttt{n,np}}\ \textbf{ice}^{\texttt{NN}}_{\texttt{n}})\ \textbf{fall}^{\texttt{VBG}}_{\texttt{vp}_{ng}}\right)\ [\textbf{somebody}^{\texttt{DT}}_{\texttt{n}}]_{\texttt{np}} \tag{19c}$$

applying (10) to $[\textbf{somebody}]_{\texttt{np}}$ $\Downarrow$

$$\textbf{which}^{\texttt{WDT}}_{\texttt{vp,np,np}}\ \left(\textbf{on}^{\texttt{IN}}_{\texttt{np,vp}_{ng},\texttt{vp}_{ng}}\ (\textbf{a}^{\texttt{DT}}_{\texttt{n,np}}\ \textbf{ice}^{\texttt{NN}}_{\texttt{n}})\ \textbf{fall}^{\texttt{VBG}}_{\texttt{vp}_{ng}}\right)\ (\textbf{a}^{\texttt{DT}}_{\texttt{n,np}}\ \textbf{person}^{\texttt{NN}}_{\texttt{n}}) \tag{19d}$$

We have presented rewriting rules which explain three most frequent type-changes $\texttt{n} \mapsto \texttt{np}$, $\texttt{vp} \mapsto (\texttt{n}, \texttt{n})$ and $\texttt{vp} \mapsto (\texttt{np}, \texttt{np})$ used by the CCG parsers. The rules are able to eliminate almost all type-changes in the sentences of FraCaS (Cooper et al., 1996) and the training portion of SICK (Marelli et al., 2014b).[22]

3.4.4 Fixing wrong analyses

After explaining most of type-changes in CCG terms, it is easier to correct several erroneous syntactic analyses coming from the CCG parsers. Most wrong analyses that can be corrected automatically are related to NPs. The main source of these errors is CCGbank. Since the sentences in CCGbank were semi-automatically obtained from phrase-structure trees, some analyses that required manual inspection were treated in s default manner. For example, all relative clauses in CCGbank are attached to NPs, i.e. all relative clauses are treated as non-restrictive modifiers. Later, Honnibal et al. (2010) added restrictivity distinctions in rebanked CCGbank. All $NP\backslash NP$ modifiers that are not preceded by punctuation were changed into $N\backslash N$ and moved inside the NP. Despite this, the CCG parsers trained on rebanked CCGbank still make mistakes with respect to adnominal attachments.

Since the RTE datasets which we will use later (see §6.1) do not contain non-restrictive modifiers, we change the $NP\backslash NP$ category of adnominals into $N\backslash N$. The rule (20) puts a relative clause under a determiner, and therefore attaches the clause directly to a noun constituent. In other words, the rule transforms the NP-S analysis of relative clause into the Nom-S analysis. This transformation is important as the Nom-S analysis offers compositional semantics of noun phrases in a straightforward manner (Partee, 1975).

$$\boldsymbol{w}^{\texttt{WP}|\texttt{WDT}}_{\texttt{vp,np,np}}\ \boldsymbol{V}_{\texttt{vp}}\ (\boldsymbol{D}_{\texttt{n,np}}\ \boldsymbol{N}_{\texttt{n}}) \rightsquigarrow \boldsymbol{D}\ (\boldsymbol{w}_{\texttt{vp,n,n}}\ \boldsymbol{V}\ \boldsymbol{N}) \tag{20}$$

$$\textbf{which}^{\texttt{WDT}}_{\texttt{vp,np,np}}\ \textbf{run}_{\texttt{vp}}\ (\textbf{every}_{\texttt{n,np}}\ \textbf{girl}_{\texttt{n}}) \rightsquigarrow \textbf{every}_{\texttt{n,np}}\ (\textbf{which}^{\texttt{WDT}}_{\texttt{vp,n,n}}\ \textbf{run}_{\texttt{vp}}\ \textbf{girl}_{\texttt{n}}) \tag{20a}$$

The correction in (20a) demonstrates (20) in action. Notice that in (20a) the result of (18b) is modified. This shows how (20) can further operate on the terms that were already processed by (18). The rule can also correct results of the rule (13).

Attaching a prepositional phrase to a noun, instead of an NP, is done by (21). This transformation allows more transparent compositional semantics for a restrictive modifier, similarly to the Nom-S analysis.

$$\boldsymbol{p}^{\texttt{IN}}_{\texttt{np,np,np}}\ \boldsymbol{T}_{\texttt{np}}\ (\boldsymbol{D}_{\texttt{n,np}}\ \boldsymbol{N}_{\texttt{n}}) \rightsquigarrow \boldsymbol{D}\ (\boldsymbol{p}_{\texttt{np,n,n}}\ \boldsymbol{T}\ \boldsymbol{N}) \tag{21}$$

[22] These datasets are discussed in §6.1.

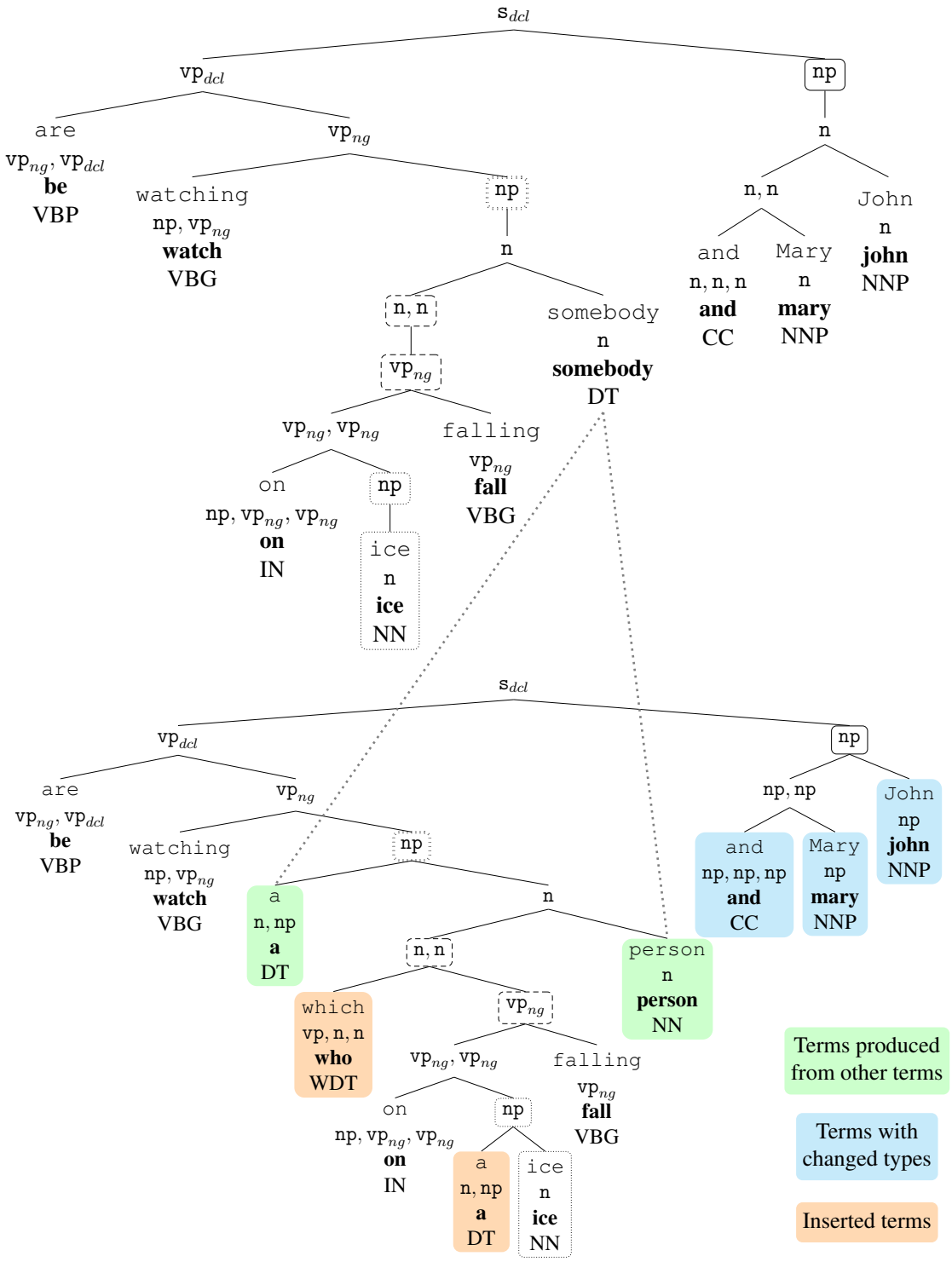


Figure 3.10: Above is a CCG term obtained from a CCG tree for *"John and Mary are watching somebody falling on ice"* produced by rebanked C&C. Below is λ-term obtained from the CCG tree using correction rules described in this section.

$$\mathbf{in}^{\texttt{IN}}_{\texttt{np,np,np}}\ \mathbf{Tbilisi}_{\texttt{np}}\ (\mathbf{a}_{\texttt{n,np}}\ \mathbf{hotel}_{\texttt{n}}) \rightsquigarrow \mathbf{a}_{\texttt{n,np}}\ (\mathbf{in}^{\texttt{IN}}_{\texttt{vp,n,n}}\ \mathbf{Tbilisi}_{\texttt{np}}\ \mathbf{hotel}_{\texttt{n}}) \tag{21a}$$

The CCG parsers often attach a relative clause directly to the first noun on its left side, e.g., *"a good (boxer who is a father)$_N$"*. This kind of analysis makes difficult to capture a constituent *"good boxer"* and gives raise to the wrong entailment of *"a good father"*. Rebanked CCGbank explicitly encourages such analyses because some adjectives, such as "*present*" and "*former*", require scope over the qualified noun (Honnibal et al., 2010). Unfortunately, such adjectives occur rarely in FraCaS and SICK. Due to this reason, we introduce the rule (22) which moves a relative clause from the head noun to the qualified noun; see (22a).

$$\boldsymbol{A}^{\texttt{HeadPOS:JJ}}_{\texttt{n,n}}\ (\boldsymbol{w}_{\texttt{vp,n,n}}\ \boldsymbol{V}_{\texttt{vp}}\ \boldsymbol{N}_{\texttt{n}}) \rightsquigarrow \boldsymbol{w}\ \boldsymbol{V}\ (\boldsymbol{A}\ \boldsymbol{N}) \tag{22}$$

$$\mathbf{good}^{\texttt{JJ}}_{\texttt{n,n}}\ (\mathbf{which}_{\texttt{vp,n,n}}\ \mathbf{is\_a\_father}_{\texttt{vp}}\ \mathbf{boxer}_{\texttt{n}}) \rightsquigarrow \mathbf{which\ is\_a\_father}\ (\mathbf{good\ boxer}) \tag{22a}$$

We have described the correction procedure which consists of three parts: (i) simplification of terms, e.g., identifying several MWEs; (ii) elimination of type-changes, e.g., explaining the most common type-change $\texttt{n} \mapsto \texttt{np}$; and (iii) fixing the analysis related to adnominals. After applying the correction rules to CCG terms, more semantically adequate CCG terms, often free from type-changes, are obtained for the sentences in FraCaS and SICK. A demonstration of the whole procedure is shown on a non-trivial example in Figure 3.10, where the final (i.e. corrected) CCG term contains no type-changes. The pseudocode of the entire correcting procedure can be found in algorithm 1 (Appendix B).

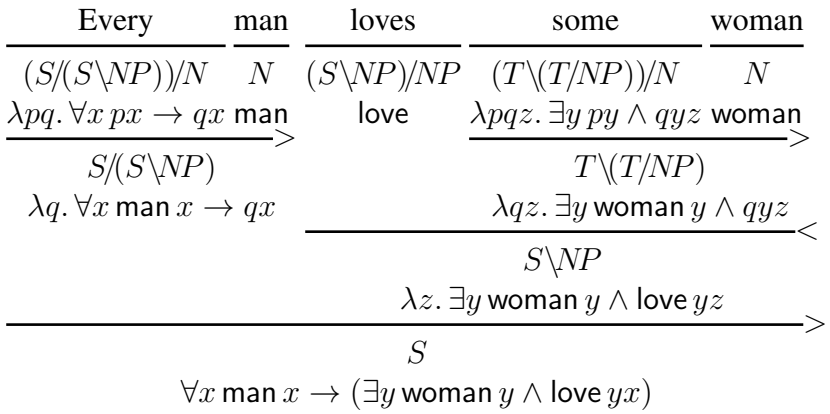
3.5 Type-raising quantifiers

There is the last step left in order to obtain LLFs from corrected CCG terms. This step involves conversion of quantified NPs like "*every man*", "*most women*" and "*exactly three dogs*" into generalized quantifiers (GQs) (Montague, 1973; Barwise and Cooper, 1981). GQs are crucial for our approach as they capture monotonic features of quantifiers which are pivotal for monotonicity reasoning (van Benthem, 1986; Sánchez-Valencia, 1991), discussed in § 1.3. In § 1.3 and § 2.1, we have also demonstrated how monotonicity reasoning facilitates proofs in the natural tableau system. In order to treat quantified NPs as GQs, we type-raise the np type to $(\texttt{vp}, \texttt{s})$, where vp abbreviates $(\texttt{np}, \texttt{s})$. More specifically, this is achieved by *type-raising* the type of a quantifier, e.g., $\mathbf{every}_{\texttt{n,np}}\ \mathbf{man}_{\texttt{n}} \rightsquigarrow \mathbf{every}_{\texttt{n,vp,s}}\ \mathbf{man}_{\texttt{n}}$.[23] In general, the type-raising procedure is not deterministic and might result in several LLFs differing in terms of order of quantifiers—having all quantifier scopes resolved.[24] First, we discuss GQs and a quantifier scope ambiguity in the CCG formalism and the CCG parsers. Then, we describe a type-raising procedure, which represents an improved version of the nested Cooper storage (Keller, 1988; Cooper, 1983).

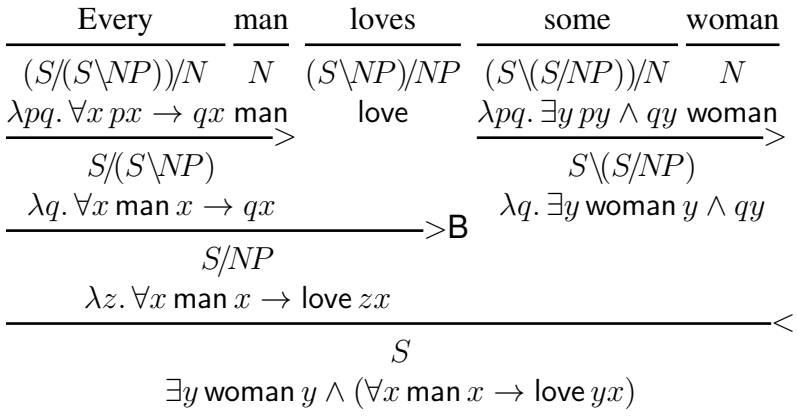
In categorial grammars, quantified NPs are usually treated as generalized quantifiers (GQs) since this enables compositional derivation of several semantic readings conditioned by a scope ambiguity. In CCG, quantifiers are usually of category $(T/(T\backslash NP))/N$

[23] Strictly speaking, changing $\texttt{n}, \texttt{np}$ into $\texttt{n}, \texttt{vp}, \texttt{s}$ is not type-raising, but we still call it so since the value type np of $\texttt{n}, \texttt{np}$ undergoes type-raising.

[24] For an overview of problems and formal approaches to quantifier scope, we refer readers to Ruys and Winter (2011).



(a) *Every man* with a wide scope



(b) *Some woman* with a wide scope

Figure 3.11: Resolving quantifier scope ambiguity for a sentence *"every man loves some woman"*. Logical forms with different quantifier scopes are obtained from different CCG derivations. Note the different lexical semantics of *some* in each derivation.

or $(T\backslash(T/NP))/N$, where T is a meta-variable. After applying quantifiers to a common noun of category N, the result is of category $T/(T\backslash NP)$ or $T\backslash(T/NP)$, respectively. These are the same categories as of type-raised NPs (see (>T) and (<T)). Therefore, quantifiers of these categories are type-raised, opposed to a non-typed-raised ones of category NP/N.

In Figure 3.11, an example shows how CCG can capture semantic readings caused by quantifier scope ambiguity with the help of type-raised quantifiers. Note that the subject quantifier "*every*" maintains its lexical category and semantics in both derivations while the object quantifier "*some*" has different lexical categories and semantics. Put differently, the semantics of a quantifier depends on the syntactic function of the noun phrase. This is somewhat unappealing treatment of quantifiers in CCG.[25]

Yet another straightforward way of modeling quantifiers in CCG is to assign them NP/N category and map NP to $(et)t$. This leads to uniform types $(et)t$ and $(et)(et)t$ for logical forms of noun phrases and quantifiers, respectively (unlike the former ap-

[25] In addition, CCG is not able to derive a semantic reading with wide-scope quantification from a medial position. For instance, it is not possible to obtain semantics of "*someone introduced everyone to John*" where "*everyone*" takes a wide-scope: for everybody there is someone who introduced him/her to John.

proach, where only quantified noun phrases have a logical form of type $(et)t$). Mapping NP to $(et)t$ also complicates lexical semantics for several lexical items. For example, in this case a transitive verb has two logical forms $\lambda OS.\, S(\lambda y.\, O(\lambda x.\, \mathsf{love}\, x\, y))$ and $\lambda OS.\, O(\lambda x.\, S\ \mathsf{love}\, x))$ of type $\big(e(et)\big)\big(e(et)\big)t$ corresponding to subject and object wide scope readings, respectively.

The CCG parsers and CCGbank are created with efficient parsing in mind, which means that treating quantifiers as type-raised is not an optimal solution: this treatment often complicates derivations and requires more lexical categories for describing quantifiers (e.g., different categories for a subject and an object quantifiers as shown in Figure 3.11). Therefore, the parsers opt for the second approach and assign NP/N category to quantifiers, and also to determiners. Since the parsers are only responsible for syntactic derivations (i.e. derivations without any logical form), the ambiguity in lexical semantics caused by the second approach has nothing to do with parsing efficiency. The issue of ambiguity is then left to those who aim to get logical forms from the derivations.

In order to type-raise quantifiers in a CCG term, first we decompose a syntactic tree of a CCG term into parts and then several trees can be constructed back from these parts. Reconstructed trees represent CCG terms with type-raised quantifiers. The procedure affects noun phrases formed from a determiner and a common noun while proper names are type-raised only in certain cases, e.g., in conjunction with quantified NPs. The procedure is also robust in the sense that it can process a CCG term with type-changes in it.

The type-raising procedure starts with chopping a syntactic tree of a CCG term of type $\mathtt{s}$. We do not chop the tree at the root and at the nodes which have λ-abstraction or the type-changing operator as a child. The tree is chopped at non-terminal nodes that are labeled with an $\mathtt{np}$ type or a syntactic type having $\mathtt{s}$ as a final value, denoted by $(-, \mathtt{s})$; for example, $\mathtt{vp}$ is such kind of type. Chopping at nodes of type $(-, \mathtt{s})$, enables to type-raise a quantified NP in its local clause. For the demonstration, the chopped parts (a–d) of the CCG term (23a) are given in Figure 3.12.

Every man who ate a burger died (23)

$$\mathbf{die}_{\mathtt{vp}}\ \big(\mathbf{every}_{\mathtt{n},\mathtt{np}}\big(\mathbf{who}_{\mathtt{vp},\mathtt{n},\mathtt{n}}\ (\mathbf{eat}_{\mathtt{np},\mathtt{vp}}\ (\mathbf{a}_{\mathtt{n},\mathtt{np}}\ \mathbf{burger}_{\mathtt{n}}))\ \mathbf{man}_{\mathtt{n}}\big)\big) \tag{23a}$$

Both copies of each chopped node are labeled with a fresh variable. The name sharing maintains reference between them. One copy is always a terminal node in a source part (i.e. serves as a free variable in a source sub-term) and another one becomes a root of a chopped branch. After the tree is chopped into pieces, reconstruction of a new tree starts from the part which has the root node of the initial tree, e.g., (a) in Figure 3.12. During reconstruction cut branches are glued back to the main part. The gluing procedure is not deterministic as there are several choice points while attaching the branches. Below we give four rules for gluing branches.

The rules instruct how the main part can further be developed depending on its structure. While doing so the following notation will be used. Each cut branch, i.e. a sub-tree or a sub-term, is represented as $X_\alpha\{\vec{u}\}$ where X is a variable label for the root (in uppercase), α is the type of the root, and $\{\vec{u}\}$ is a set of free variables on the sub-tree, where $\vec{u}$ is a comma separated (possibly empty) sequence. For the rules, we will employ the vector representation for terms and types (see §2.0.1). We also denote a sub-tree by its root label X, in short. For instance, the sub-tree in Figure 3.12b can be denoted by $X_{\mathtt{np}}\{y\}$ or by X shortly.

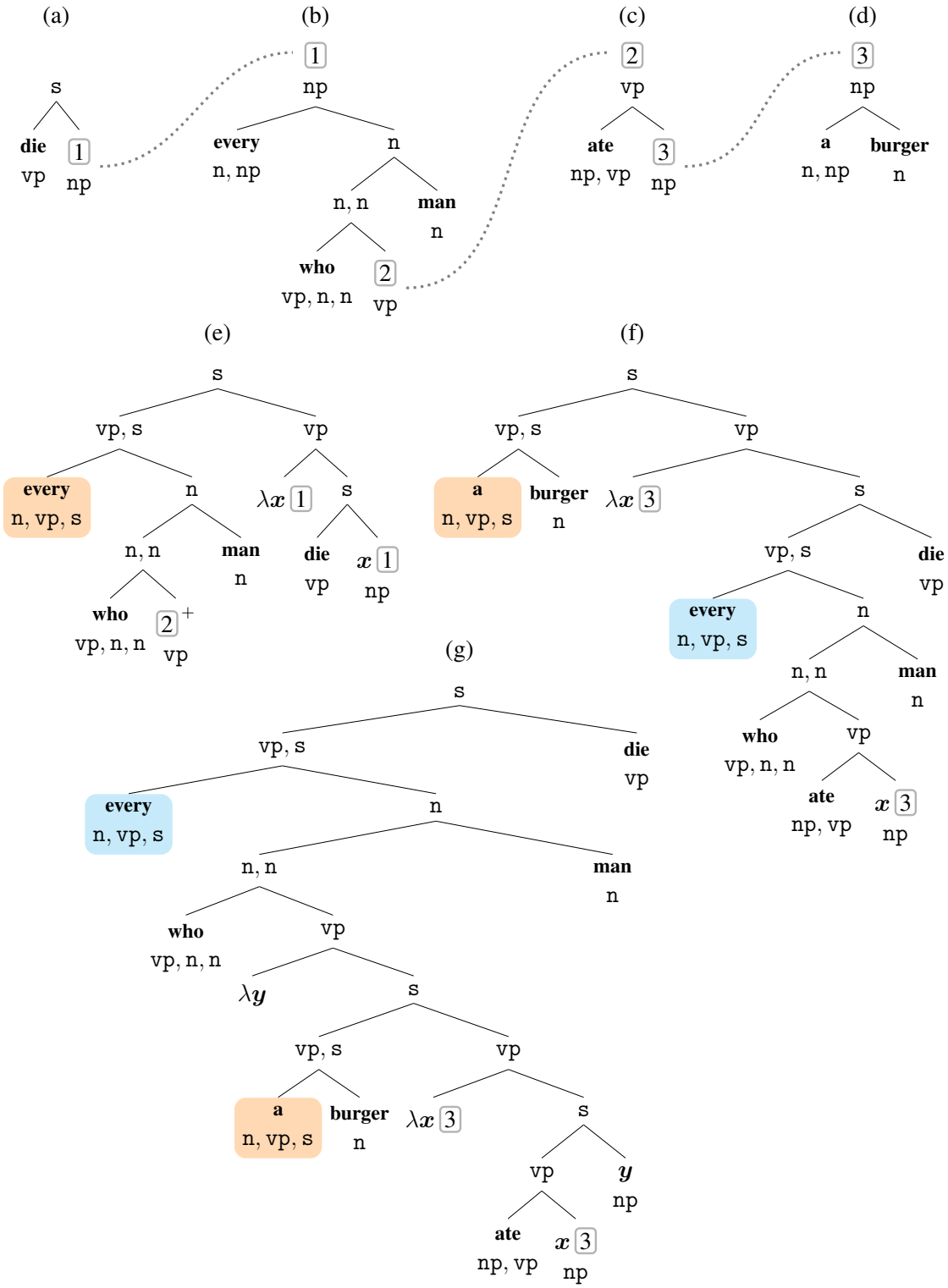


Figure 3.12: A syntactic tree of the CCG term (23a) chopped into 4 parts (a–d). The reconstruction starts from attaching (b) to (a) and type-raising $\textbf{every}_{\text{n,np}}$ according to (DTR); the result is (e). Since a free variable, denoted by 2, is marked in (e), there are two ways to plug (c) in (e), either according to (NTR) or (WTR); results for each case are (f) and (g), respectively.

The *default type-raising* (DTR) rule operates on the main three of type $\overrightarrow{\alpha}\mathtt{s}$ that has at least one free variable of type np. The rule application is non-deterministic if there are more than one free variables of type np. For some selected free variable x, the rule feeds the main tree X with fresh variables $\overrightarrow{v}_{\overrightarrow{\alpha}}$ until it becomes of type s and then abstracts x from it. The type-raised version of the sub-term X is later applied to the latter term.[26] In the end, introduced fresh variables are abstracted back.

$$T_{\overrightarrow{\alpha}\mathtt{s}}\{x_{\mathtt{np}}, \overrightarrow{y}\} \;\Rightarrow\; \lambda\overrightarrow{v}.\,\mathtt{TR}(X)(\lambda x.\,T^{+}\overrightarrow{v}_{\overrightarrow{\alpha}}) \tag{DTR}$$

In the resulted term, the free variables in T are marked with $+$ meaning that they are in a scope of some quantifier. This information about the scope will be used later to allow different scope orders. The (DTR) rule is used in Figure 3.12, when (e) is obtained from the main part (a).

The *substitution* rule allows to attach a sub-term X of type $(-,\mathtt{s})$ to a main term. The restriction on the sub-term is that it has no free variables of type np. The sub-term is directly plugged in the main term by substituting the corresponding free variable x. After the substitution, the sign g of x is distributed over the free variables of X.

$$T_{(-,\mathtt{s})}\{x^{g}_{(-,\mathtt{s})}, \overrightarrow{y}\} \quad \text{where} \quad X\{\overrightarrow{z} \mid u_{\mathtt{np}} \notin \overrightarrow{z}\} \;\Rightarrow\; T[X^{g}/x] \tag{SUB}$$

The *narrow type-raising* (NTR) rule is similar to (SUB), but in contrast to it, (NTR) glues back the sub-term that contains a free variable of type np. Before plugging the sub-term X in the variable x, first the sub-term of type np corresponding to one of the free variables of X is type-raised according to (DTR). Put differently, the term of a free-variable of X is type-raised in a local clause.

$$T_{(-,\mathtt{s})}\{x^{g}_{(-,\mathtt{s})}, \overrightarrow{y}\} \quad \text{where} \quad X\{\overrightarrow{z} \mid u_{\mathtt{np}} \in \overrightarrow{z}\} \;\Rightarrow\; T[\mathtt{DTR}^{g}(X)/x] \tag{NTR}$$

For example, in Figure 3.12, (g) is obtained by attaching (c) to (e) with the help of (NTR).

The previous rule allows type-raising of an NP term of X with a narrow scope. But if the variable x occurs under the scope of some quantifier, then the *wide scope-raising* (WTR) makes it possible to type-raise an NP term of X with a wide scope.

$$T_{\overrightarrow{\alpha}\mathtt{s}}\{x^{+}_{(-,\mathtt{s})}, \overrightarrow{y}\} \;\text{ where }\; X\{\overrightarrow{z} \mid u_{\mathtt{np}} \in \overrightarrow{z}\} \;\Rightarrow\; \lambda\overrightarrow{v}.\,\mathtt{TR}(U)(\lambda u.\,T[X^{+}/x]\overrightarrow{v}_{\overrightarrow{\alpha}}) \tag{WTR}$$

Notice that the application of (WTR) is allowed if an NP term of X happens to be under the scope of some quantifier. On the other hand (NTR) is applicable despite this condition. In Figure 3.12, (WTR) is applied to (e) and (c) and in order to obtain (f). The process of type-raising quantifiers for the CCG term in (23a) is shown in Figure 3.12, where the final reconstructed trees (f) and (g) represent the LLFs found in (23c) and (23b), respectively.

$$\mathbf{EVERY}_{\mathtt{n,vp,s}}\big(\mathbf{who}\ \big(\lambda\boldsymbol{x}.\,\mathbf{A}_{\mathtt{n,vp,s}}\ \mathbf{burger}\ (\lambda\boldsymbol{y}.\,\mathbf{eat}\ \boldsymbol{y}_{\mathtt{np}}\ \boldsymbol{x}_{\mathtt{np}})\big)\mathbf{man}\big)\mathbf{die} \tag{23b}$$

$$\mathbf{A}_{\mathtt{n,vp,s}}\ \mathbf{burger}\big(\lambda\boldsymbol{y}.\,\mathbf{EVERY}_{\mathtt{n,vp,s}}\big(\mathbf{who}\ (\lambda\boldsymbol{x}.\,\mathbf{eat}\ \boldsymbol{y}_{\mathtt{np}}\ \boldsymbol{x}_{\mathtt{np}})\mathbf{man}\big)\mathbf{die}\big) \tag{23c}$$

[26] TR is a type-raising operator that takes a term of type np and returns a term of type $((\mathtt{np},\mathtt{s}),\mathtt{s})$. This is done by tracing the leftmost branch of the term in order to find the main lexical function term. After the term is found its type is changed correspondingly. The operator takes into account the coordination and modifier terms.

From a procedural perspective, the presented four rules can be applied to the main sub-term in different orders. Some application orders result the same LLF while other orders produce the LLFs with different quantifier orders. There are only two types of choice point in the presented rules. One is related to (DTR) when choosing a variable of type np for type-raising. Another choice point is made when choosing the rule between (NTR) and (WTR). Decision at these points is crucial for an order of quantifiers.

For a sentential coordination like (24a), the procedure of type-raising is deterministic as none of the two choice points occur during rule applications. The obtained LLF is given in (24b). The alternative LLFs, (24d) and (24c), are semantically equivalent to (24b) but they push the conjunction deep in a structure. From the viewpoint of tableau theorem proving, these two LLFs have inefficient structures since the rules for quantifiers should be applied prior to the rule for a conjunction. Notice that the LLFs in (24d) and (24c) are prevented with the help of (NTR), which type-raises quantified NPs in a local clause if they are not under some quantifier scope. By identifying local clauses, the approach makes an initial attempt to detect scope islands. Future research in this direction is required to make the account more feasible.

Every man sleeps and some woman worries (24)

$\mathbf{and}_{\mathtt{s},\mathtt{s},\mathtt{s}}\big(\mathbf{sleep}_{\mathtt{vp}}(\mathbf{every}_{\mathtt{n},\mathtt{np}}\ \mathbf{man}_{\mathtt{n}})\big)\big(\mathbf{worry}_{\mathtt{vp}}(\mathbf{some}_{\mathtt{n},\mathtt{np}}\ \mathbf{woman}_{\mathtt{n}})\big)$ (24a)

$\mathbf{and}\ (\mathbf{EVERY}_{\mathtt{n},\mathtt{vp},\mathtt{s}}\ \mathbf{man}\ \mathbf{sleep})(\mathbf{SOME}_{\mathtt{n},\mathtt{vp},\mathtt{s}}\ \mathbf{woman}\ \mathbf{worry})$ (24b)

$\mathbf{SOME}_{\mathtt{n},\mathtt{vp},\mathtt{s}}\mathbf{woman}\big(\lambda \boldsymbol{y}.\ \mathbf{EVERY}_{\mathtt{n},\mathtt{vp},\mathtt{s}}\mathbf{man}\big(\lambda \boldsymbol{x}.\ \mathbf{and}\ (\mathbf{sleep}\boldsymbol{x})(\mathbf{worry}\ \boldsymbol{y})\big)\big)$ (24c)

$\mathbf{EVERY}_{\mathtt{n},\mathtt{vp},\mathtt{s}}\mathbf{man}\big(\lambda \boldsymbol{x}.\ \mathbf{SOME}_{\mathtt{n},\mathtt{vp},\mathtt{s}}\mathbf{woman}\big(\lambda \boldsymbol{y}.\ \mathbf{and}\ (\mathbf{sleep}\boldsymbol{x})(\mathbf{worry}\ \boldsymbol{y})\big)\big)$ (24d)

The described type-raising procedure shares similarities with the nested Cooper storage (Keller, 1988) technique if chopped branches are seen as nested pairs of a variable and a term. The original Cooper storage (Cooper, 1983) produces a storage from a syntactic tree and lexical semantics, and then several semantic readings are reproduced by retrieving elements from the storage in different ways. The nested Cooper storage (Keller, 1988) improves the original work by allowing a hierarchy in a storage and accounting for nested NPs in this way. In order to get certain scope orderings, sometimes one needs to start reconstructing a semantic reading from different nested storages. Our proposed approach differs from the storage approaches in several aspects. Unlike them, our approach operates on a tree structure without any additional lexical semantics. We get all our readings from the same set of chopped branches. We achieve this with four reconstruction rules, in contrast to their single rule. During reconstruction, our approach takes into account local clauses.

The current type-raising procedure is not complete in the sense that it is not able to generate all possible semantic readings. This shortcoming comes from syntactic types of CCG terms rather than from the procedure itself. The reason is that syntactic types in CCG terms are less flexible for type-raising. For instance, with syntactic types it is possible only one semantic reading for a CCG term (25a), in particular (25b), where *"a pub"* takes scope over *"every man"*, i.e. there is a pub in which every man is drunk. In case of semantic types, both readings (25c) and (25d) corresponding to two orders of quantifiers are available.[27] (25c) is similar to (25b) while (25d) is the uncaptured reading

[27] In semantic terms, the type p abbreviates et type, i.e. a property type.

with a meaning that every man in *any* pub is drunk. Our type-raising procedure works also for CCG terms with semantic types if $\mathtt{np}$ and $(-,\mathtt{s})$ types are replaced with e and $(-,t)$, respectively.

Every man in a pub is drunk (25)

$\mathbf{is}_{\mathrm{vp}_{adj},\mathrm{vp}}\ \mathbf{drunk}_{\mathrm{vp}_{adj}}\big(\mathbf{every}_{\mathrm{n},\mathrm{np}}\ (\mathbf{in}_{\mathrm{np},\mathrm{n},\mathrm{n}}\ (\mathbf{a}_{\mathrm{n},\mathrm{np}}\ \mathbf{pub}_{\mathrm{n}})\ \mathbf{man}_{\mathrm{n}})\big)$ (25a)

$(\mathbf{A}_{\mathrm{n},\mathrm{vp},\mathrm{s}}\ \mathbf{pub}_{\mathrm{n}})\ \lambda\boldsymbol{x}.\ \mathbf{EVERY}_{\mathrm{n},\mathrm{vp},\mathrm{s}}(\mathbf{in}_{\mathrm{np},\mathrm{n},\mathrm{n}}\ \boldsymbol{x}_{\mathrm{np}}\ \mathbf{man}_{\mathrm{n}})(\mathbf{is}_{\mathrm{vp}_{adj},\mathrm{vp}}\ \mathbf{drunk}_{\mathrm{vp}_{adj}})$ (25b)

$(\mathtt{A}_{ppt}\ \mathtt{pub}_{p})\ \lambda x.\ \mathtt{EVERY}_{ppt}\ (\mathtt{in}_{epp}\ x_e\ \mathtt{man}_{p})(\mathtt{is}_{pp}\ \mathtt{drunk}_{p})$ (25c)

$\mathtt{EVERY}_{ppt}\big(\lambda y.\ \mathtt{A}_{ppt}\ \mathtt{pub}_{p}(\lambda x.\ \mathtt{in}_{epp}\ x_e\ \mathtt{man}_{p}\ y_e)\big)\big(\mathtt{is}_{pp}\ \mathtt{drunk}_{p}\big)$ (25d)

This shortcoming of syntactic types is not very significant while dealing with textual entailments. For example, we could not find the problem in FraCaS and SICK that is sensitive to the order of quantifiers. But if one still wants to consider more semantic readings of a sentence, then it is one step away from the current approach. More specifically, a CCG term with semantic types can be obtained by translating syntactic types into semantic ones, and then applying the same algorithm of type-raising to the CCG term will result more semantic readings.

We have discussed type-raising of quantified NPs in the context of CCG and showed how a quantifier scope ambiguity is resolved there. Since we do not want to complicate lexical semantics and types in LLFs, e.g., NP being a subtype of $(et)t$, we opt for the procedure that reconstructs λ-terms. In particular, we have presented the type-raising procedure which first decomposes a CCG term into linked pieces and then reconstructs possibly several LLFs from them. The reconstruction is guided by four rules. While sub-terms are combined, the rules also type-raise NP terms. Due to several choice points in reconstruction, as a result several LLFs with different quantifier orders are obtained. The process of reconstruction is depicted in Figure 3.12. In general, the procedure is not able to produce all possible LLFs because syntactic types appear to be less flexible than semantic ones. Despite this, the type-raising procedure produces most of LLFs. Obtained LLFs have efficient structures as type-raising with a narrow scope is carried out in a local clause: (24b) vs. (24c) and (24d).

3.6 Conclusion

In the chapter we have described the procedure how to obtain LLFs automatically from wide-coverage natural language expressions. Since categorial grammar, like LLFs, model lexical elements as functions, we start from the CCG derivation trees produced by the state-of-the-art CCG parsers. The first processing step is to discard directionality from syntactic categories and combinatory rules. As a result, from a CCG derivation we obtain a structure called a CCG term. Later, a CCG term is corrected through three subprocedures consisting of rewriting rules. First, the lexical entries are simplified via reducing them to canonical forms and several MWEs are identified as a lexical term. Second, the type-changes, which are heavily used by the parsers, are eliminated to a great extent. Third, we employ rules to fix some wrong syntactic analyses related to relative clauses and PP attachments, e.g., to move relative clauses and PP attachments from NPs

to Nouns. The latter change makes the terms semantically more adequate. The rewriting rules employed in the correction procedure is collected while exploring FraCaS and the trial portion of SICK (see § 6.2). Notice that the correction process is facilitated by the absence of word order in CCG terms. The last step to obtain LLFs is to type-raise quantified NPs in corrected CCG terms. The type-raising procedure chops a CCG term and rebuilds several LLFs from it. The number of LLFs is conditioned by the quantifier scope ambiguity. The pipeline that converts a CCG derivation into a list of LLFs hereafter will be referred as the LLF generator.[28]

The LLF generator is a useful NLP tool as it produces adequate logical forms from wide-coverage CCG derivations. Its final product, LLFs, can be seen as abstract semantic representations. They represent semantic composition trees and are valuable structures for the research on open-domain compositional semantics. One can express linguistic semantics in his/her favorite semantic representation language by assigning corresponding lexical semantics to lexical terms in an LLF. For example, it is obvious how to obtain first-order logic representations or Discourse Representation Structure (DRS) (Kamp and Reyle, 1993) from LLFs: it is sufficient to annotate lexical terms with λ-terms for first-order logic or with λ-DRSs (Muskens, 1996; Bos, 2008).

Recently the number of applications of CCG parsers for wide-coverage semantic analysis is rapidly increasing. Shortcomings of CCG derivations are usually fixed in semantic lexicons. This means that if one wants to use fixed CCG derivations, either he/she has to opt for specific semantic representations or fix the derivations for his/her own semantic representation. Advantage of an LLF is that it separates the fixing procedure and the derivation of semantics from each other. Furthermore, it is obvious that any application of CCG derivations for wide-coverage semantics is transferable on LLFs.

After knowing how the LLFs for wide-coverage text looks like, now we can design tableau rules for LLFs that encode wide range of syntactic constructions. In the next chapter, we present a plethora of such rules.

[28] At the current state, we did not model the scope ambiguity caused by negation. The reason is that the phenomenon does not appear crucial when solving the FraCaS and SICK textual entailments with the natural tableau system. We leave this issue for future work.

Appendix B

Algorithm 1: Pseudocode of the recursive procedure which fixes CCG terms.

input: a CCG term $\boldsymbol{T}$;
case *Rule* is applicable to $\boldsymbol{T}$:
  (9) $[\boldsymbol{c}_{\mathrm{n}}^{\mathtt{NNP|NNPS|PRP}}]_{\mathrm{np}} \rightsquigarrow \boldsymbol{c}_{\mathrm{np}}$: apply it to $\boldsymbol{T}$; **go to** 2;
  (10) $[\boldsymbol{c}_{\mathrm{n}}^{\mathtt{DT}}]_{\mathrm{np}} \rightsquigarrow \boldsymbol{d}_{\mathrm{n,np}}^{\mathtt{DT}}\ \boldsymbol{n}_{\mathrm{np}}^{\mathtt{NN}}$ where $\mathtt{split}(\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{n})$: apply it to $\boldsymbol{T}$; **go to** 2;
  (11) $[\boldsymbol{c}_{\mathrm{n,n}}^{\mathtt{DT|CD}}\ \boldsymbol{N}_{\mathrm{n}}]_{\mathrm{np}} \rightsquigarrow \boldsymbol{c}_{\mathrm{n,np}}\ \boldsymbol{N}$: apply it to $\boldsymbol{T}$; **go to** 2;
  (12) $[\boldsymbol{e}_{(\mathrm{n,n}),\mathrm{n,n}}^{\mathtt{RB}}\ \boldsymbol{c}_{\mathrm{n,n}}^{\mathtt{CD}}\ \boldsymbol{N}_{\mathrm{n}}]_{\mathrm{np}} \rightsquigarrow \boldsymbol{e}_{(\mathrm{n,np}),\mathrm{n,np}}\ \boldsymbol{c}_{\mathrm{n,np}}\ \boldsymbol{N}_{\mathrm{n}}$: apply it to $\boldsymbol{T}$; **go to** 2;
  (13) $\left[[\boldsymbol{V}_{\mathrm{vp}}]_{\mathrm{n,n}}\ \boldsymbol{T}_{\mathrm{n}}\right]_{\mathrm{np}} \rightsquigarrow [\boldsymbol{V}_{\mathrm{vp}}]_{\mathrm{np,np}}\ [\boldsymbol{T}_{\mathrm{n}}]_{\mathrm{np}}$: apply it to $\boldsymbol{T}$; **go to** 2;
  (14) $[\boldsymbol{c}_{\mathrm{n,n,n}}\ \boldsymbol{X}\ \boldsymbol{Y}]_{\mathrm{np}} \rightsquigarrow \boldsymbol{c}_{\mathrm{np,np,np}}\ [\boldsymbol{X}]_{\mathrm{np}}\ [\boldsymbol{Y}]_{\mathrm{np}}$: apply it to $\boldsymbol{T}$; **go to** 2;
  (15) $[\boldsymbol{T}_{\mathrm{n}}^{\mathtt{HeadPOS:NN}}]_{\mathrm{np}} \rightsquigarrow \mathbf{a}_{\mathrm{n,np}}\ \boldsymbol{T}$: apply it to $\boldsymbol{T}$; **go to** 2;
  (16) $[\boldsymbol{T}_{\mathrm{n}}^{\mathtt{HeadPOS:NNS}}]_{\mathrm{np}} \rightsquigarrow \mathbf{s}_{\mathrm{n,np}}\ \boldsymbol{T}^{\mathtt{HeadPOS:NN}}$: apply it to $\boldsymbol{T}$; **go to** 2;
  (17) $[\boldsymbol{V}_{\mathrm{vp}}]_{\mathrm{n,n}}\ \boldsymbol{N}_{\mathrm{n}} \rightsquigarrow \mathbf{which}_{\mathrm{vp,n,n}}\ \boldsymbol{V}_{\mathrm{vp}}\ \boldsymbol{N}$: apply it to $\boldsymbol{T}$; **go to** 2;
  (18) $[\boldsymbol{V}_{\mathrm{vp}}]_{\mathrm{np,np}}\ \boldsymbol{T}_{\mathrm{np}} \rightsquigarrow \mathbf{which}_{\mathrm{vp,np,np}}\ \boldsymbol{V}_{\mathrm{vp}}\ \boldsymbol{T}_{\mathrm{np}}$: apply it to $\boldsymbol{T}$; **go to** 2;
case *Rule* is applicable to $\boldsymbol{T}$:
  (20) $\boldsymbol{w}_{\mathrm{vp,np,np}}^{\mathtt{WP|WDT}}\ \boldsymbol{V}_{\mathrm{vp}}\ (\boldsymbol{D}_{\mathrm{n,np}}\ \boldsymbol{N}_{\mathrm{n}}) \rightsquigarrow \boldsymbol{D}\ (\boldsymbol{w}_{\mathrm{vp,n,n}}\ \boldsymbol{V}\ \boldsymbol{N})$: apply it to $\boldsymbol{T}$; **go to** 13;
  (21) $\boldsymbol{p}_{\mathrm{np,np,np}}^{\mathtt{IN}}\ \boldsymbol{T}_{\mathrm{np}}\ (\boldsymbol{D}_{\mathrm{n,np}}\ \boldsymbol{N}_{\mathrm{n}}) \rightsquigarrow \boldsymbol{D}\ (\boldsymbol{p}_{\mathrm{np,n,n}}\ \boldsymbol{T}\ \boldsymbol{N})$: apply it to $\boldsymbol{T}$; **go to** 13;
  (22) $\boldsymbol{A}_{\mathrm{n,n}}^{\mathtt{HeadPOS:JJ}}\ (\boldsymbol{w}_{\mathrm{vp,n,n}}\ \boldsymbol{V}_{\mathrm{vp}}\ \boldsymbol{N}_{\mathrm{n}}) \rightsquigarrow \boldsymbol{w}\ \boldsymbol{V}\ (\boldsymbol{A}\ \boldsymbol{N})$: apply it to $\boldsymbol{T}$; **go to** 13;
case *Term* matches to $\boldsymbol{T}$:
  $\boldsymbol{x}$: $\boldsymbol{S} := \boldsymbol{T}$;
  $\boldsymbol{c}$: $\boldsymbol{S} := \boldsymbol{T}$;
  $\boldsymbol{F}\ \boldsymbol{A}$: $\boldsymbol{S} := \mathtt{fixCCG}(\boldsymbol{F})\ \mathtt{fixCCG}(\boldsymbol{A})$;
  $\lambda \boldsymbol{x}.\ \boldsymbol{F}$: $\boldsymbol{S} := \lambda \boldsymbol{x}.\ \mathtt{fixCCG}(\boldsymbol{F})$;
  $[\boldsymbol{F}_{\alpha}]_{\beta}$: $\boldsymbol{S} := [\mathtt{fixCCG}(\boldsymbol{F})]_{\alpha}$;
return $\boldsymbol{S}$;

Combinatory rule	C&C rule in Prolog notation	Transformation schema
Forward functional application	$\texttt{fa}(Y, \mathbf{F}_{Y/X}, \mathbf{A}_X)$	$\mathbf{F}_{(\mathtt{x},\mathtt{y})}\ \mathbf{A}_{\mathtt{x}}$
Backward functional application	$\texttt{ba}(Y, \mathbf{A}_X, \mathbf{F}_{Y\backslash X})$	
Type-changing rule (i.e. lexical rule)	$\texttt{lx}(Y, X, \mathbf{A}_X)$	$[\mathbf{A}_{\mathtt{x}}]_{\mathtt{y}}$
Conjunction	$\texttt{conj}(X\backslash X, X, \mathbf{C}_{conj}, \mathbf{A}_X)$	$\mathbf{C}_{\mathtt{x},\mathtt{x},\mathtt{x}}\ \mathbf{A}_{\mathtt{x}}$
Forward type raising	$\texttt{tr}\big(Y/(Y\backslash X), \mathbf{A}_X\big)$	$\lambda \boldsymbol{f}_{(\mathtt{x},\mathtt{y})}.\ \boldsymbol{f}_{(\mathtt{x},\mathtt{y})}\ \mathbf{A}_{\mathtt{x}}$
Backward type raising	$\texttt{tr}\big(Y\backslash(Y/X), \mathbf{A}_X\big)$	
Forward functional composition	$\texttt{fc}(Z/X, \mathbf{F}_{Z/Y}, \mathbf{G}_{Y/X})$	$\lambda \boldsymbol{v}_{\mathtt{x}}.\ \mathbf{F}_{(\mathtt{y},\mathtt{z})}\ (\mathbf{G}_{\mathtt{x},\mathtt{y}}\ \boldsymbol{v}_{\mathtt{x}})$
Backward functional composition	$\texttt{fc}(Z\backslash X, \mathbf{G}_{Y\backslash X}, \mathbf{F}_{Z\backslash Y})$	
Forward crossing functional composition	$\texttt{fxc}(Z\backslash X, \mathbf{F}_{Z/Y}, \mathbf{G}_{Y\backslash X})$	
Backward crossing functional composition	$\texttt{bxc}(Z/X, \mathbf{G}_{Y/X}, \mathbf{F}_{Z\backslash Y})$	
Generalized-2 forward functional composition	$\texttt{gfc}\big((W/Y)/X, \mathbf{F}_{W/Z}, \mathbf{G}_{(Z/Y)/X}\big)$	$\lambda \boldsymbol{v}_{\mathtt{x}} \boldsymbol{u}_{\mathtt{y}}.\ \mathbf{F}_{(\mathtt{z},\mathtt{w})}\ (\mathbf{G}_{\mathtt{x},(\mathtt{y},\mathtt{z})}\ \boldsymbol{v}_{\mathtt{x}}\ \boldsymbol{u}_{\mathtt{y}})$
Generalized-2 backward functional composition	$\texttt{gbc}\big((W\backslash Y)\backslash X, \mathbf{G}_{(Z\backslash Y)\backslash X}, \mathbf{F}_{W\backslash Z}\big)$	
Generalized-2 forward crossing functional composition	$\texttt{gfxc}\big((W\backslash Y)\backslash X, \mathbf{F}_{W/Z}, \mathbf{G}_{(Z\backslash Y)\backslash X}\big)$	
Generalized-2 backward crossing functional composition	$\texttt{gbxc}\big((W/Y)/X, \mathbf{G}_{(Z/Y)/X}, \mathbf{F}_{W\backslash Z}\big)$	
Right punctuation	$\texttt{rp}(X, \mathbf{A}_X, \mathbf{P}_{punct})$	$\mathbf{A}_{\mathtt{x}}$
Left punctuation	$\texttt{lp}(X, \mathbf{P}_{punct}, \mathbf{A}_X)$	
Right punctuation type-changing	$\texttt{rtc}(Y, \mathbf{A}_X, \mathbf{P}_{punct})$	$[\mathbf{A}_{\mathtt{x}}]_{\mathtt{y}}$
Left punctuation type changing	$\texttt{ltc}(Y, \mathbf{P}_{punct}, \mathbf{A}_X)$	

Table 3.2: Transformations for the CCG combinatory rules found in the C&C parser. A Prolog representation of the rules is one of the supported output formats in the C&C parser. Syntactic categories and types of constituents are written in a subscript. Due to a type changing operator $[\,.\,]_\alpha$, obtained structures are not identical but similar to typed λ-terms.

Chapter 4

Inventory of Tableau Rules

In the previous chapters we have tuned the natural tableau system for wide-coverage natural reasoning and presented the method for obtaining Lambda Logical Forms (LLFs). Now, in this chapter, it is time to show how to reason with the tableau system over the obtained LLFs. Since, in general, reasoning capacity of a tableau system depends on the inventory of tableau rules, we will present a plethora of tableau rules that account for various lexical and phrasal semantics.

Tableau rules are inference rules that decompose a formula and express its semantics in terms of its sub-formulas. More diverse *semantic constructions* are available in logic more tableau rules are needed to decompose these constructions. When we talk about semantic constructions, for instance, in propositional case we distinguish $P \wedge Q$ from $P \vee Q$. Though they have an similar structure, where a binary predicate applies to arguments, as semantic constructions they differ from each other because $\wedge$ and $\vee$ have different semantics. So, constants with different predefined semantics can trigger different semantic constructions.

If we think about natural language as a formal logic, then each lexical item comes with its own predefined semantics. Roughly speaking, does this mean that each lexical item will need its own tableau rule? We could do so, but fortunately there are other more optimal ways. In particular, semantic relations over lexical items, e.g., **dog** $\sqsubseteq$ **animal**, are pushed in a Knowledge Base (KB). This enables to model the semantic relations with the help of a few tableau rules. But still, in order to reduce semantic relations over phrases to the relations over lexical items, we need tableau rules that decompose phrases. Here, we should be ready that at least several dozens of rules will be required for this.

The goal of the chapter is exactly this—to present a plethora of tableau rules that model semantics of various phrases. The rules are inspired by the textual entailment problems found in the SICK (Marelli et al., 2014b) and FraCaS (Cooper et al., 1996) datasets.[1] Several rules that model specific syntactic phenomena are somewhat biased towards the analyses provided by the Combinatory Categorial Grammar (CCG) parsers. The collected set of rules cover constructions involving prepositional phrases (PPs), definite NPs, adjectives, passives, auxiliaries, phrases with the copula, expletives, open noun compounds, light verb constructions, passives, etc. The rules are distributed in sections according to the phenomena they are applicable to. But since there are rules that concern more than one phenomenon at the same time, these rule are presented only once in one of

[1] See §6.2.1 for more details about the collection procedure of tableau rules.

the relevant sections. Each presented rule is supported with a textual entailment example mostly drawn from the explored RTE datasets.

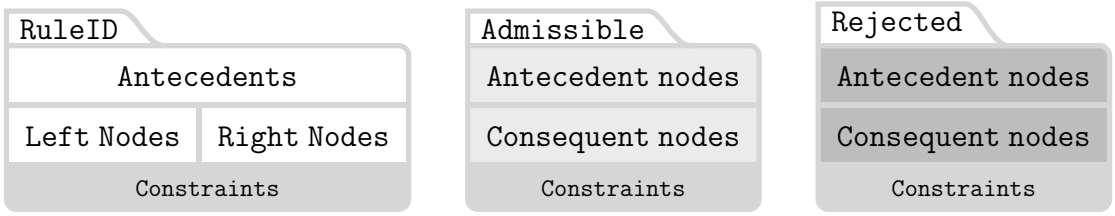
While our aim is to collect the tableau rules that account for a wide range of linguistic constructions, we are not interested to have a complete tableau system. In other words, we prioritize an incomplete proof system for a larger fragment of natural logic rather than a complete system for a smaller fragment. Additionally, we target to have sound tableau rules which will allow to have as few wrong proofs as possible.

The rest of the chapter is structured as follows. First, we present rules for modifier terms, involving adjectives and auxiliaries. Then rules for prepositional phrases are introduced. Some of those rules have functions beyond semantic analysis: they try to surmount the problem related to PP attachments. We sketch several approaches for modeling definite NPs and choose the one which treats them as referring expressions. Closure rules are designed to account for the expletive "*there*", verb subcategorization (in an efficient way), open compound nouns and light verbs. Apart from some additions, rules for the copula resemble the tableau rules for equality. Rules for passive constructions and attitude verbs are the last rules to be discussed.

4.0 Preliminaries

Several conventions will be assumed while presenting tableau rules and proofs in the consequent sections. Most of these conventions were already mentioned in § 2.0, but for completeness of the chapter, we briefly mention them here too.

The tableau rules are presented in a *folder format*. Antecedent nodes are above the horizontal line while consequent nodes are below it. In case of branching rule, consequent branches are delimited by a vertical line. The name of a tableau rule is written above. A possible set of constraints over the entries is written below, in a gray area. We will use three types of style to indicate the status of a tableau rule. Rules with a transparent background are ordinary rules. Rules with a light gray background are *admissible*—in presence of some (ordinary) rules, they are redundant from completeness point of view. Rejected rules which are given for the sake of demonstration are with a dark gray background.



The following conventions are made with respect to terms. Uppercase meta-variables match any term; lowercase meta-variables match only constant terms; $\vec{C}$ and $\vec{c}$ match possibly an empty sequence of terms and constants respectively; $\mathbb{X}$ matches either $\mathbb{T}$ or $\mathbb{F}$ while $\overline{\mathbb{X}}$ stands for a negation of $\mathbb{X}$, where $\overline{\mathbb{T}} = \mathbb{F}$ and $\overline{\mathbb{F}} = \mathbb{T}$. We use the extended format of tableau entries (18). An empty list of arguments or modifiers is denoted by [ ]. In order to have leaner tableau entries, empty modifier lists are omitted.

$$\underbrace{\texttt{memoryList} : \texttt{LLF} : \texttt{argumentList}}_{\text{ternary format of a term}} : \texttt{truthSign} \tag{18}$$

Types written as a subscript of a term act as powerful constraints on terms. Like in case of terms, several assumptions are also made with respect to types. The lowercase Greek letters are used as meta-variables over types. The type $(\mathtt{np}, \mathtt{s})$, corresponding to VPs, is abbreviated as $\mathtt{vp}$. We employ the vector representation from §2.0.1. For instance, $\vec{\alpha}, \mathtt{s}$ type represents the sentential type. It can match the types like $\mathtt{s}$, $\mathtt{vp}$ and $(\mathtt{pp}, \mathtt{np}, \mathtt{vp})$. Often we will put constraints on the final type of a term. $(-, \alpha)$ denotes a type with α as its final (i.e. the most right) type; for instance, $(-, \mathtt{s})$ can match $\mathtt{s}$, $(\mathtt{np}, \mathtt{s})$ or $(\mathtt{vp}, \mathtt{vp})$, where the latter two can also be matched by $(-, \mathtt{vp})$. Notice that $(-, \alpha)$ matches $\vec{\gamma}\alpha$ but the former avoids redundant type variables. We also employ Kleene star and Kleene plus to denote the non-empty sequence or the possibly empty sequence of the same types. For example, $\mathtt{np}^{+}\mathtt{s}$ matches $\mathtt{vp}$ and $(\mathtt{np}, \mathtt{vp})$ but not $(\mathtt{pp}, \mathtt{vp})$. We write $\vec{C}_{\vec{\alpha}}$, where the lengths of the sequences are the same, if the terms in $\vec{C}$ are of type the elements of $\vec{\alpha}$ respecting the order. Sometimes the types of terms are omitted if they are irrelevant for a discussion, but usually the omitted types can be inferred from the context.

Similarly to the rewriting rules of §3.4, in tableau rules, meta-variables ranging over terms may be further restricted with the information provided by the parsers (e.g., POS or NER tags); this information is usually written as a superscript. The algebraic properties of a term can also be written in a superscript. The terms might be represented by the same string of characters and still be of different types, in this case these terms are considered different. If in a tableau rule the same term is mentioned several times, only one of them is annotated with its type and others are assumed to be of the same type.

4.1 Rules for modifiers

In natural language sentences, it is quite common that a word or phrase modifies another word or phrase. In the theories of syntax, the former constituent is referred as an *adjunct* or a *modifier* and the latter one as a *head*.[2] The modifiers are usually optional and their omission in a phrase does not influence on its grammatical status. The number of modifiers per head can vary in contrast to complements, where the number of complements are strictly defined via an argument structure of a head. In this section, we present tableau rules for modifier terms like auxiliaries and adjectives. The rules will complement the other rules for modifiers already presented in §2.3.2 and §2.4.1.

4.1.1 Rules for auxiliaries

A main function of auxiliary verbs is to express tense and aspect. On the other hand, textual entailment problems found in Recognizing Textual Entailment (RTE) datasets are rarely sensitive to tense and aspect. Usually tense and aspect are explicitly ignored by the annotation guidelines of RTE datasets (Dagan et al., 2006). Furthermore, they are irrelevant for the datasets we use for collecting tableau rules. Due to these reasons, in the current version of the natural tableau, we do not model tense and aspect. Therefore, the auxiliary verbs, like "*be*" or "*do*", that contribute to aspect or tense marking are ignored

[2] Throughout this work the concepts of an adjunct and a modifier are considered synonyms while some theories of syntax do not use them synonymously. Similarly, we also do not distinguish arguments and complements from each other.

in LLFs.[3] This is done with the help of the (AUX) rule. The rule simply identifies such lexical terms (they are usually a modifier of a VP) and discards them. In other words, these terms are interpreted as identity functions.

AUX

$$\frac{[\overrightarrow{M}] : a_{\text{vp,vp}}\ V : [C] : \mathbb{X}}{[\overrightarrow{M}] : V : [C] : \mathbb{X}}$$

$a \in \{\textbf{be}, \textbf{do}, \textbf{have}, \textbf{will}\}$

The rule is one of the frequently used rules since the auxiliaries occur in negative and passive constructions that are plenty in the RTE datasets. Moreover, the default tense in the SICK problems is present progressive meaning that there is always a lexical constant **be** in the sentences.

The rule in action is demonstrated in several tableau proofs, including Figure 4.10 and Figure 4.11 in Appendix C. In the implemented tableau system (Chapter 5) the scope of the rule is extended and (AUX) is also used to discard, for instance, "*to*" from infinitives and "*that*" from embedded clauses.

4.1.2 Rules for adjectives

We give several rules that model the properties usually associated with adjectives. These are the subsective, intersective and privative properties that are familiar in the works since late 1960's (Kamp and Partee, 1995). The *subsective* and *intersective* properties of modifiers were already defined in Definition 12 (§ 2.4). It is straightforward to account for these properties via tableau rules.

The rule ($\subset_{\mathbb{T}}$) models the subsective property of a modifier. It simply discards a subsective modifier in a true context. The rule automatically works for the subsective adjectives (and PPs) when $\alpha = \mathtt{n}$.[4] The rule allows a possibly empty modifier list with certain properties on its members. After introducing ($\subset_{\mathbb{T}}$), we can consider ($\subset \mathrm{M}_{\mathbb{T}}$) in §2.4.1 as a shortcut for the applications of ($\subset_{\mathbb{T}}$) and (M>). With the help of ($\subset_{\mathbb{T}}$), subsective modifiers are discarded without putting them in a modifier list. Now, if we consider a famous example in PROB-7, which requires proper treatment of the subsective adjective "*skillful*", then the natural tableau system will correctly classify it as neutral.

$\subset_{\mathbb{T}}$

$$\frac{[\overrightarrow{M}] : A^{\subset}_{\alpha,\alpha} H : [\overrightarrow{C}] : \mathbb{T}}{[\overrightarrow{M}] : H : [\overrightarrow{C}] : \mathbb{T}}$$

$\forall E \in \overrightarrow{M}\ (A \rightleftharpoons E)$ or $\overrightarrow{M}^{\uparrow}$

Mary is a violinist and a *skillful* surgeon
Mary is a skillful violinist

PROB-7

[3] Another evidence of neglecting the tense is demonstrated by representing lexical terms with lemmata. Hence based on the term representation, it is not possible to distinguish the word forms of the same verb that are in the present and past simple tenses.

[4] The subsective adjectives are also referred as *standard* (Parsons, 1972) or *affirmative* (Cooper et al., 1996).

As we already mentioned in § 2.4.1, intersective terms are automatically subsective. This makes ($\subset_{\mathbb{T}}$) applicable to intersective terms, but the rule cannot completely unfold semantics of intersective terms. For this reason we give more specific rules: ($\cap_{\mathbb{T}}$) and ($\cap_{\mathbb{F}}$). To show that a term is intersective, we mark it with $\cap$ in a superscript.[5] Notice the type-coercion in ($\cap_{\mathbb{T}}$): it introduces a new lexical term of type et for the intersective adjective. Put differently, the intersective adjectives can be seen as dual where the term of syntactic type models the syntactic part while the term of semantic type carries semantics. When the construction is in a false context, ($\cap_{\mathbb{F}}$) branches. The type-coercion also takes place in this rule. In both rules, if there is a non-empty modifier list, it is copied to the node that contains the head noun.[6]

$\cap_{\mathbb{T}}$

$$[\overrightarrow{M}] : a^{\cap}_{\mathrm{n,n}}\ N : [c_e] : \mathbb{T}$$

$$[\overrightarrow{M}] : N : [c] : \mathbb{T} \qquad a_{et} : [c] : \mathbb{T}$$

$$\forall E \in \overrightarrow{M}\ (a \rightleftharpoons E)$$

$\cap_{\mathbb{F}}$

$$[\overrightarrow{M}] : a^{\cap}_{\mathrm{n,n}}\ N : [c_e] : \mathbb{F}$$

$$[\overrightarrow{M}] : N : [c] : \mathbb{F} \qquad a_{et} : [c] : \mathbb{F}$$

$$\forall E \in \overrightarrow{M}\ (a \rightleftharpoons E)$$

$-_{\mathbb{T}}$

$$a^{-}_{\mathrm{n,n}}\ N : [c_e] : \mathbb{T}$$

$$N : [c] : \mathbb{F}$$

There are also adjectives that do not satisfy the subsective property. For example, "*apparent*", "*alleged*" and "*possible*" do not satisfy the subsective property since, for example, being "*apparent success*" does not entail being "*success*". Such adjectives are called *non-subsective*. An interesting subcategory of non-subsective adjectives is *privative*. The adjective $a_{\mathrm{n,n}}$ is privative if and only if it satisfies the property in (1). Usually the adjectives like "*false*", "*imaginary*", "*fake*", "*former*" and "*fictitious*" are considered to be privative, but the consideration is uncontroversial.[7]

$$\forall N_{\mathrm{n}}(a_{\mathrm{n,n}}N \sqsubseteq -N) \tag{1}$$

In spite of the unclear status of privative adjectives, we can still account for the privative property (1) in the tableau system. The rule ($-_{\mathbb{T}}$) captures this property. The rule is useful at least for those entailment problems that presuppose the privative property for certain adjectives, e.g., FraCaS-198 from the FraCaS data (Cooper et al., 1996; MacCartney and Manning, 2007). In this way, the textual entailment problem is proved as contradiction with the help of ($-_{\mathbb{T}}$).

[5] There is no need to annotate the intersective adjectives with both $\subset$ and $\cap$ signs as after the application of ($\cap_{\mathbb{T}}$) there is no need for applying ($\subset_{\mathbb{T}}$).

[6] The commuting constraints on modifier list members will be relaxed in the implemented tableau prover. We will assume that subsective and intersective modifiers commute, which is not actually true strictly speaking. Consider **high-paying**$^{\subset}_{\mathrm{n,n}}$**part-time**$^{\cap}_{\mathrm{n,n}}$**job**$_{\mathrm{n}}$ and **part-time**$^{\cap}_{\mathrm{n,n}}$**high-paying**$^{\subset}_{\mathrm{n,n}}$**job**$_{\mathrm{n}}$. The former does not necessarily mean "*high-paying job*" while the latter entails this meaning.

[7] For example, somebody who is "*former president*" can still be "*president*" after selecting second time. Also interpretability of well-formed questions like "*is this gun real or fake?*" begs a question whether the class of privative adjectives is itself imaginary. One of the solutions, offered by Partee (2001), is to consider a privative adjective as subsective while shifting the meaning of the head noun. For instance, "*fake gun*" or "*stone lion*" are understood after "*gun*" and "*lion*" extend their meanings to include things having the corresponding shape.

John is a former university student
John is a university student
GOLD: cont; FraCaS-198

We presented the rules that model the most well known properties of adjectives: subsective, intersective and privative. The intuition behind the rules is quite simple as the properties are themselves simple. While the existence of the privative adjectives is questionable, from the completeness point of view, we still presented the rule for the privative property.

4.2 Rules for prepositions

Prepositions are heavily used in unrestricted natural language text.[8] Therefore it is very important to account for them in the wide-coverage tableau system. Prepositions usually represent a head of a Prepositional Phrase (PP), where a PP itself can be an argument or a modifier of a noun or verb phrase. Hence the rules for prepositions are expected to deal with a wide range of constructions.

We start the section with introducing the problem related to *PP attachments* since, as it is shown later, several rules for prepositions are significantly influenced by this problem. Then the new rules for prepositions are introduced along with their supporting examples. The rules are presented in three groups as the rules of each group account for the similar phenomenon, in particular: the modifier PPs, a nature of PP attachment and a site of PP attachment. Some of the rules are unsound and the only reason for adopting them is to overcome the mistakes in PP analyses made by the parsers. Throughout the section, we assume transitive prepositions, i.e. those that form and head PPs, while talking about prepositions. In the last subsection, this convention is dropped as we discuss the rules for the transitive and intransitive prepositions, the latter often referred as *particles*. There, the term "*preposition*" refers to both types of prepositions.

4.2.1 The problem of PP attachment

In natural language sentences, a PP often brings structural or semantic ambiguity. It can be ambiguous in terms of the site where the PP is attached to and in terms of a nature of the attachment. For example, based on the Part of Speech (POS) tags of the words the sentences in (2–5) are equivalent but differ from each other in terms of a site or a nature of PP attachments. The sites of PP attachments are different in (2) and (3). The PP "*with his hands*" in (2) represents an optional argument of the VP: "*his hands*" is in the instrumental thematic relation with the VP. On the other hand, the PP "*with Sam*" in (3) modifies the VP. The difference in the nature of PP attachment is demonstrated in (4) and (5), where the PPs are a modifier and an argument, respectively, for the same noun.

$$\text{John } \big[\text{[ate a roll]}_{VP/PP} \text{ [with his hands]}_{PP}\big]_{VP} \tag{2}$$

[8] Baldwin et al. (2009) reports that four out of the top-ten most frequent words in the British National Corpus are prepositions.

John $\big[$[ate a roll]$_{VP}$ [with Sam]$_{VP\backslash VP}\big]_{VP}$ (3)

John ate a $\big[$roll$_{N}$ [with eel]$_{N\backslash N}\big]_{N}$ (4)

John ate a $\big[$roll$_{N/PP}$ [of sushi]$_{PP}\big]_{N}$ (5)

The ambiguity associated with PP attachment makes difficult to correctly analyze the constructions with PPs. This problem is known as *PP attachment resolution* or *PP attachment disambiguation* and is regarded as a challenging task in syntactic parsing. Usually the problems under these names are more specific and aim to resolve only the ambiguity related to a site of PP attachment.[9] PP attachment resolution as a four-way classification task (Merlo and Ferrer, 2006), demonstrated in (2–5), is less common compared to its binary counterpart.

John ate a $\big[$[roll]$_{N}$ [with his hands]$_{N\backslash N}\big]_{N}$ (2a)

John $\big[$[ate a roll]$_{VP/PP}$ [with Sam]$_{PP}\big]_{VP}$ (3a)

John $\big[$[ate a roll]$_{VP}$ [with eel]$_{VP\backslash VP}\big]_{VP}$ (4a)

Similarly to other syntactic parsers, the CCG parsers are also imperfect in the analysis of PPs. The CCG parsers are based on the lexicalized grammar formalism that makes them sensitive to both dimensions, site and nature, of PP attachments. They might attach a PP to a wrong phrase, like in (2a) or (4a), or attach it to a correct phrase but in a wrong way, like in (3a).[10] These kinds of mistake in the parse trees are further projected in LLFs and they may prevent the tableau system from the correct decision. For instance, after wrongly identifying "*with eel*" as a VP modifier in (4a), the entailment relation vanishes between the LLFs obtained from (4) and (4a).

The rules that we present in the following subsections try not only to unfold the semantics of the phrases with PPs but also to abstract from the mistakes made by the parsers while analyzing PP attachments.

4.2.2 Rules for prepositional phrases

The tableau rules for prepositions are divided into three groups. The first group contains the rules that are broadly sound for correct LLFs—assuming that the CCG parsers make no mistakes while analyzing PPs. Since it is a hard problem to identify and correct the wrong PP attachments made by the CCG parsers, we design the tableau rules that surmount this shortcoming of the parsers. In particular, the second and third groups consist of the rules that overcome the mistakes made by the parsers with respect to PP attachments. The mistakes involve both types of errors, in terms of a site and a nature of PP attachment. The presented rules are supported with the concrete examples from the RTE datasets.

[9] The RRR dataset by Ratnaparkhi et al. (1994) is a standard benchmark for PP attachment disambiguation. It consists of quadruples of words $\langle verb, noun_1, preposition, noun_2\rangle$ where a system should decide whether the PP formed by $preposition$ and $noun_2$ is attached to $verb$ or to $noun_1$. Notice that the task is simplified as only head words are included in a quadruple in contrast to the whole phrases. This binary simplification of the problem is connected with the phrase structure grammar which usually neglects the relation between adjacent constituents. As a result, the differences between the analyses of (2) and (3) or (4) and (5) are erased.

[10] In the analyses similar to (2a–4a), it is said that *semantic-selectional restrictions* are violated: although the sentences are grammatical, they have nonsensical meanings.

4.2.2.1 Rules for modifier PPs

We start with presenting the rules for the PPs that act as modifiers, e.g., the PPs in (3) and (4). Such PPs are syntactically optional and behave in similar way as intersective adjectives or adverbs. Therefore new rules for modifier PPs are expected to be similar to the rules for intersective terms.

The rules (PP@N$_{\mathbb{T}}$) and (PP@N$_{\mathbb{F}}$) treat a PP term as an intersective modifier for nouns. The lexical term p in the antecedent entries of the rules is required to be a preposition (constrained with the POS tag IN in its superscript) of type $(\mathtt{np}, \mathtt{n}, \mathtt{n})$. The consequent entries contain the same prepositional term p but with different $(\mathtt{np}, \mathtt{pp})$ type, which itself is a subtype of eet, i.e. a type of binary relation between entities. Preference to the syntactic type $(\mathtt{np}, \mathtt{pp})$ over the semantic eet is supported by the fact that $(\mathtt{np}, \mathtt{pp})$ is another common type of the prepositional terms in the constructions like (2) and (5). In this way, $(\mathtt{np}, \mathtt{pp})$ can be a canonical type for prepositions instead of the syntactically less informative eet type.

PP@N$_{\mathbb{T}}$

$$[\overrightarrow{M}] : p^{\mathtt{IN}}_{\mathtt{np},\mathtt{n},\mathtt{n}} d\ N : [c] : \mathbb{T}$$

$$[\overrightarrow{M}] : N : [c] : \mathbb{T} \qquad p_{\mathtt{np},\mathtt{pp}} : [d, c] : \mathbb{T}$$

$$\forall E \in \overrightarrow{M}\ (p^{\mathtt{IN}}_{\mathtt{np},\mathtt{n},\mathtt{n}} d \rightleftarrows_N E)$$

PP@N$_{\mathbb{F}}$

$$[\overrightarrow{M}] : p^{\mathtt{IN}}_{\mathtt{np},\mathtt{n},\mathtt{n}} d\ N : [c] : \mathbb{F}$$

$$[\overrightarrow{M}] : N : [c] : \mathbb{F} \qquad p_{\mathtt{np},\mathtt{pp}} : [d, c] : \mathbb{F}$$

$$\forall E \in \overrightarrow{M}\ (p^{\mathtt{IN}}_{\mathtt{np},\mathtt{n},\mathtt{n}} d \rightleftarrows_N E)$$

The rules can also apply to the tableau entries with the non-empty modifier list. Soundness of the rules is supported by the assumption that the PP term $p^{\mathtt{IN}}_{\mathtt{np},\mathtt{n},\mathtt{n}} d$ commutes with each term in the modifier list with respect to N. Based on this assumption, the modifier list is further copied with the head term N in the consequent entries.

Let us consider the tableau that searches a model for a sentence "*John ate huge sushi with avocado*". In the tableau, there will be a node asserting that there is some entity c that is "*huge sushi with* d", where d is "*avocado*". Potentially the parsers can return at least two different analysis of the phrase: (a) "*huge*" modifies "*sushi*" first, and (b) "*with* d" modifies "*sushi*" first (see Figure 4.1). But in both cases, it is entailed that c is "*huge sushi*" and c is "*with* d" where d is "*avocado*". In the tableaux of Figure 4.1, this is achieved with the help of (PP@N$_{\mathbb{T}}$)'s ability to copy a modifier list to the consequent node. Handling the modifier list in such a way, makes the tableau system more immune towards the mistakes made by the parsers.

Adverbial PPs (i.e. the PPs that modify a VP) can be processed with the rules (M<), (M>) and (EV$_{\mathbb{T}}$) already presented in § 2.3. Assuming that adverbials (including PPs) are subsective and commute with each other, then it is possible to separately assert them for a head verb. For example, in case of the adverbial PP this works as follows. First push the PP in a modifier list, then move it towards the head verb by swapping it with other adverbials with the help of ($\rightleftharpoons$ M) from § 2.4.1, then use ($\subset$M$_{\mathbb{T}}$) (from § 2.4.1) to discard other adverbials, and in the end pull the PP from the modifier list via (M>).

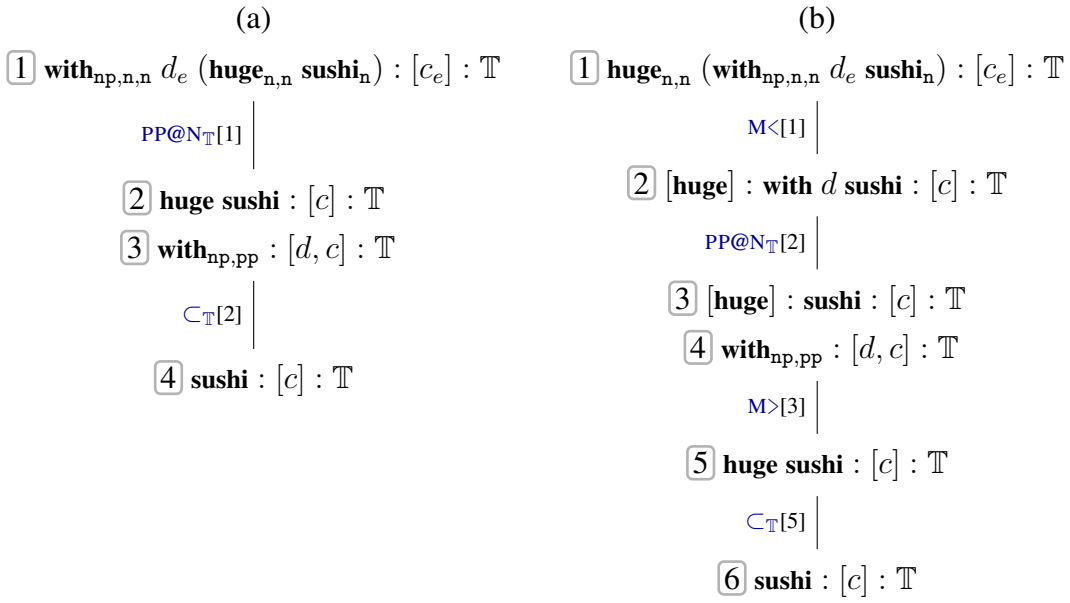


Figure 4.1: Unfolding semantics of the noun phrase "*huge sushi with avocado*"

4.2.2.2 Rules treating a nature of PP attachment

A nature of PP attachment is either an argument or a modifier with respect to the head. For NLP systems, it is hard to detect a nature of PP attachment.[11] Even human annotators can disagree on this issue. Consider the sentence in (6). FrameNet (Baker et al., 1998)—a database of semantic frames of English words—analyzes the PP "*into the office*" as a core element for a semantic frame of "*walk*". On the other hand, PropBank (Palmer et al., 2005)—the Penn Treebank annotated with predicate argument relations—counts the PP in (6) as a modifier rather than an argument.[12] It is the same case with (7): according to FrameNet the PP "*into pieces*" is an argument of the verb "*cut*" but PropBank considers it as a modifier.[13] While the argument-modifier distinction for PPs is not a settled problem, we still need to process such kind of ambiguous PPs in the wide-coverage tableau system.

Mary walked into the office at 7:30 (6)

John cut a carrot into pieces (7)

In order to model an ambiguous nature of PPs, we borrow the idea from the neo-Davidsonian approach to analyze arguments and modifiers uniformly. For example, the neo-Davidsonian analyses (6a) and (7a) of the sentences (6) and (7) do not care about the argument-modifier distinction. With the help of the thematic relations, an argument "*Mary*", a dubious argument "*office*" and a modifier "*7:30*" all are interpreted as a modifier of the event entity e.

$$\exists e\, \exists c(\mathit{walk}\, e \;\wedge\; \text{AGENT}\, \mathit{Mary}\, e \;\wedge\; \mathit{office}\, c \;\wedge\; \text{GOAL}\, c\, e \;\wedge\; \text{TIME}\, \mathit{7{:}30}\, e) \quad (6a)$$

[11] This is one of the main reasons why this aspect of PP attachment is not considered as a part of the PP attachment disambiguation challenge.

[12] The FrameNet frame for "*walk*": http://bit.ly/self_motion; the PropBank frameset for "*walk*": http://bit.ly/propbank_walk

[13] The FrameNet frame for "*cut*": http://bit.ly/framenet_cutting; the PropBank frameset for "*cut*": http://bit.ly/propbank_cut_v

$$\exists e\,\exists c(\mathit{cut}\,e \;\wedge\; \text{AGENT}\,\mathit{John}\,e \;\wedge\; \mathit{carrot}\,c \;\wedge\; \text{PATIENT}\,c\,e \;\wedge\; \text{MANNER}\,\mathit{pieces}\,e) \quad (7a)$$

Informally speaking, if we consider the prepositions as acting for the thematic relations, then all modifier and argument PPs can be seen as modifiers of the head verb. This is automatically achieved for all modifier PPs since they are of type $(\mathtt{vp}, \mathtt{vp})$. In order to treat the argument PPs as a modifier of a VP, we introduce the (V@PP) rule. The rule introduces a new tableau entry where the preposition p gets the type suitable for adverbial PPs. The tableau in Figure 4.2 shows this rule in action. It applies to 2, and in the obtained node 3 the PP term $\mathbf{into}_{\mathtt{np},\mathtt{vp},\mathtt{vp}}\ c_e$ becomes a modifier of the verb. Notice also the similarity between the last two conjuncts of the neo-Davidsonian analysis in (6a) and the nodes 8 and 5 in Figure 4.2 where the prepositions play a role of the thematic relations. After the introduction of (V@PP), now it is not crucial whether a PP, e.g. **into** c, is an argument or a modifier of a VP.

V@PP

$$[\overrightarrow{M}] : V_{\mathtt{pp},\alpha}\,(p^{\mathtt{IN}}_{\mathtt{np},\mathtt{pp}}\,D) : [\overrightarrow{C}] : \mathbb{X}$$

$$[\overrightarrow{M}] : p^{\mathtt{IN}}_{\mathtt{np},\alpha,\alpha}\,D\,V_{\alpha} : [\overrightarrow{C}] : \mathbb{X}$$

$$\alpha = (\mathtt{np}^{*}, \mathtt{vp})$$

Apart from solving the ambiguity caused by the argument-modifier distinction for PPs, the proposed approach also *cures* wrong or inconsistent analyses produced by the parsers. For example, in SICK-9069, the C&C parser gives inconsistent derivations for the similar sentences. In the premise, "*in the ocean*" is a modifier while in the conclusion "*in the water*" is an argument. Consequently the verbs also obtain different argument structures based on these analyses.[14] This mismatch makes it difficult to find a proof for the entailment relation. But the presented approach overcomes this difficulty as (V@PP) transforms the argument "*in the water*" into a modifier.

Two boys are $\big[$[laying$_{VP}$ [in the ocean]$_{VP\backslash VP}$] [close to the beach]$_{VP\backslash VP}\big]$

Two boys are $\big[$[laying$_{VP/PP}$ [in the water]$_{PP}$] [close to the beach]$_{VP\backslash VP}\big]$

C&C, GOLD: ent, SICK-9069

An ambiguity in a nature of PP attachment occurs in case of NPs too. Similarly to VPs, in the NPs a preposition can be seen as an abstract binary relation over entities. For instance, consider "*roll of sushi*" in (5). The preposition can be seen there as a relation $\mathbf{of}_{\mathtt{np},\mathtt{pp}}s_e r_e$ where r and s are "*roll*" and "*sushi*", respectively. In order to treat PP arguments of a noun as noun modifiers, we introduce (N@PP$_{\mathbb{T}}$) and (N@PP$_{\mathbb{F}}$). They process the PP arguments in the same way as (PP@N$_{\mathbb{T}}$) and (PP@N$_{\mathbb{F}}$) process PP modifiers of a noun.

[14] Though both FrameNet and PropBank agree on that the location is a core argument of "*laying*", the questions arise: which of the two locations, "*in the ocean*" and "*close to the beach*", is an argument? or do they both form a single compound argument via implicit conjunction? This issue does not pose a problem for our approach since we treat the both locative PPs uniformly as modifiers.

$$\boxed{1}\ \mathbf{at}_{\mathrm{np,vp,vp}}\ \mathbf{7{:}30}_{\mathrm{np}}\ (\mathbf{walk}_{\mathrm{pp,vp}}\ (\mathbf{into}_{\mathrm{np,pp}}\ c_e) : [\mathbf{Mary}_{\mathrm{np}}] : \mathbb{T}$$

$$\mathrm{M}{<}[1]$$

$$\boxed{2}\ [\mathbf{at\ 7{:}30}] : \mathbf{walk}_{\mathrm{pp,vp}}\ (\mathbf{into}_{\mathrm{np,pp}}\ c) : [\mathbf{Mary}] : \mathbb{T}$$

$$\mathrm{V@PP}[2]$$

$$\boxed{3}\ [\mathbf{at\ 7{:}30}] : \mathbf{into}_{\mathrm{np,vp,vp}}\ c\ \ \mathbf{walk}_{\mathrm{vp}} : [\mathbf{Mary}] : \mathbb{T}$$

$$\subset_{\mathbb{T}}[3]$$

$$\boxed{4}\ [\mathbf{at\ 7{:}30}] : \mathbf{walk}_{\mathrm{vp}} : [\mathbf{Mary}] : \mathbb{T}$$

$$\mathrm{M}{>}[4]$$

$$\boxed{5}\ \mathbf{at\ 7{:}30}\ \ \mathbf{walk}_{\mathrm{vp}} : [\mathbf{Mary}] : \mathbb{T}$$

$$\subset_{\mathbb{T}}[5]$$

$$\boxed{6}\ \mathbf{walk}_{\mathrm{vp}} : [\mathbf{Mary}] : \mathbb{T}$$

$$\subset \mathrm{M}_{\mathbb{T}}[3]$$

$$\boxed{7}\ \mathbf{into}_{\mathrm{np,vp,vp}}\ c\ \ \mathbf{walk}_{\mathrm{vp}} : [\mathbf{Mary}] : \mathbb{T}$$

Figure 4.2: Treating the PP argument of a VP as its modifier

N@PP$_{\mathbb{T}}$

$$[\overrightarrow{M}] : N_{\mathrm{pp,n}}\ P_{\mathrm{pp}} : [c_e] : \mathbb{T}$$

$$[\overrightarrow{M}] : N_{\mathrm{n}} : [c] : \mathbb{T} \qquad P : [c] : \mathbb{T}$$

$$\forall E \in \overrightarrow{M}\,(P \rightleftarrows_N E)$$

N@PP$_{\mathbb{F}}$

$$\overrightarrow{M} : N_{\mathrm{pp,n}}\ P_{\mathrm{pp}} : [c_e] : \mathbb{F}$$

$$\overrightarrow{M} : N_{\mathrm{n}} : [c] : \mathbb{F} \qquad P : [c] : \mathbb{F}$$

$$\forall E \in \overrightarrow{M}\,(P \rightleftarrows_N E)$$

Like in case of a VP (see SICK-9069), the parsers also make mistakes when analyzing the NPs with PPs. For example, the PP "*with a black bag*" in the premise of SICK-340 is incorrectly identified by the C&C parser as an argument of the noun. Despite this mistake, the entailment relation in SICK-340 is proved as (N@PP$_{\mathbb{T}}$) licenses the equivalence of **with** b **girl** and **girl** (**with** b), where b can be any entity, including "*black bag*". The tableau proof for one of the directions of the equivalence is presented in Figure 4.3.

$[\text{schoolgirl}_{N/PP}\ [\text{with a black bag}]_{PP}]$ is on a crowded train

$[\text{girl}_{N}\ [\text{with a black bag}]_{N\backslash N}]$ is on a crowded train

C&C, GOLD: `ent`, SICK-340

We have presented the rules that treat the modifier and argument PPs in the similar way—all PPs are converted into modifiers. This approach is suitable for the wide-coverage analysis and does not expect the parsers to guess a nature of PP attachment.

4.2.2.3 Rules treating the site of PP attachment

The structural ambiguity induced by PP attachments is a pervasive and, at the same time, an open problem in NLP. In order to prevent this problem hindering the tableau reasoning,

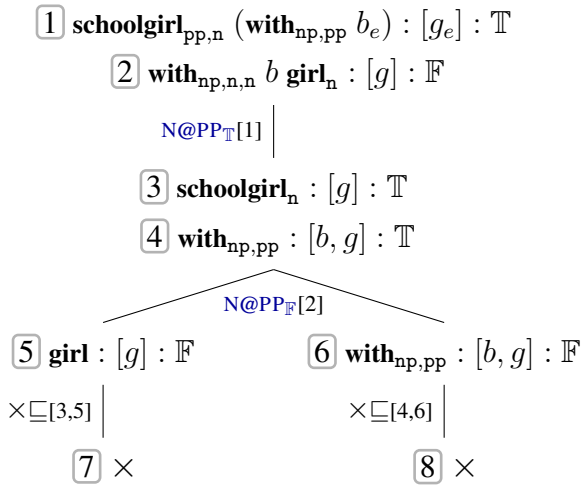


Figure 4.3: Treating a PP argument of a noun as a modifier

we propose a solution that partially abstracts reasoning from an ambiguous site of PP attachments. Thus the equivalence of (8) and (9) and the entailment of (10) from them is proved without resolving the structural ambiguity associated with the PP attachment. The tableau rules that help to achieve this are not considered as members of the core part of the tableau rules; they are optional rules that are only helpful in overcoming the mistakes made by the parsers.

A boy saw a criminal with a telescope (8)

A criminal was seen by a boy with a telescope (9)

A boy saw a criminal with some tool (10)

Incorrect resolution of the ambiguity might be crucial for recognizing textual entailment. Consider SICK-8145 which itself does not represent a difficult entailment problem—the key is to capture the local entailment between "*mit*" and "*hand*" and project it upwards. Nevertheless, after the C&C parser wrongly attaches PP_1 to "*ball*" in the conclusion, an *easily recognizable* entailment problem suddenly becomes hard to prove.[15] Another example is SICK-7064, where the correct sites of PP attachments are not straightforward. Additionally, this example also emphasizes that the PP attachment problem is not parser-bound. Independently from the parsers, there exist expressions where the correct site of PP attachments is not obvious: either it does not change semantics much, like in the sentences of SICK-7064, or the site is structurally ambiguous, as in (8–10).[16]

[15] As it was already noted, the standard task of PP attachment disambiguation represents a simplified version of the real problem. In particular, the RRR dataset (Ratnaparkhi et al., 1994) consists of extracted quadruples of head words, e.g., in case of our example it would be ⟨*has, ball, in, mitt*⟩, instead of raw phrases, e.g., *has a yellow ball in the hand*. Taking into account that the state-of-the-art systems achieve around 85% of accuracy on the *simplified* disambiguation task (Lapata and Keller, 2005), it is clear that accounting for the site issue of PP attachments is inevitable to obtain high results on the RTE task.

[16] Notice that the average human annotation for PP attachments based on the whole sentences has accuracy of 93.2% and it drops to 88.2% if only quadruples of head words, e.g. ⟨*perform, trick, on, ramp*⟩, are considered (Ratnaparkhi et al., 1994).

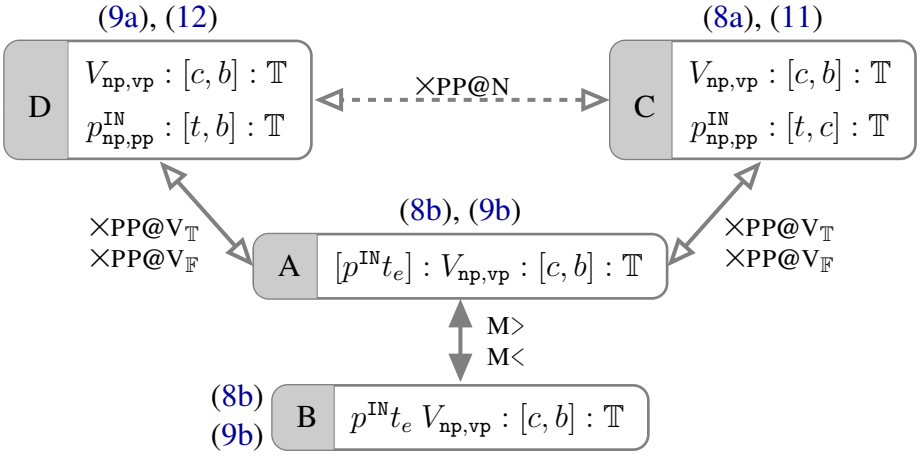


Figure 4.4: Abstracting from the site of PP attachments. The states A, B, C and D correspond to the readings caused by the structural ambiguity in PP attachments.

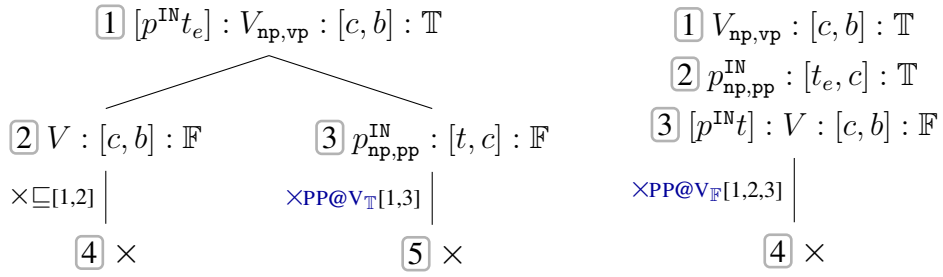
A woman in blue $\big[$[has a yellow ball]$_{VP}$ [in the mitt]$_{VP\backslash VP}\big]$
A woman in blue has a yellow $\big[$ball$_N$ [in the hand]$^1_{N\backslash N}\big]$
C&C, GOLD: ent, SICK-8145

A trick is being performed by $\big[$[a rollerblader]$_{NP}$ [on a ramp]$^1_{NP\backslash NP}\big]_{NP}$
A rollerblader is $\big[$[performing a trick]$_{VP}$ [on a ramp]$_{VP\backslash VP}\big]_{VP}$
EasyCCG, GOLD: ent, SICK-7064

Our aim is to capture the logical relations over LLFs despite the wrong site of PP attachments. In this way, the natural tableau should account for the entailment relations in SICK-7064, SICK-8145 and (8–10) despite the correct resolutions of the sites of PP attachments. As a solution to this, we suggest to treat *certain* LLFs as *paraphrases* differing from each other only in terms of the site of PP attachments: for example, to make the LLFs of (8a) and (9a) equivalent to (8b) and (9b), respectively. To achieve this, we treat the tableau entries of the form $[p^{\text{IN}} t_e] : V_{(-,\text{s})} : [\vec{A}] : \mathbb{X}$ as *underspecified representations*. In the entry it is not reliable to assume that the PP term $p^{\text{IN}} t$ modifies the verb V, because potentially the source LLF could be obtained from the wrong derivation, where the PP is wrongly attached to the VP instead of one of the arguments from $\vec{A}$.

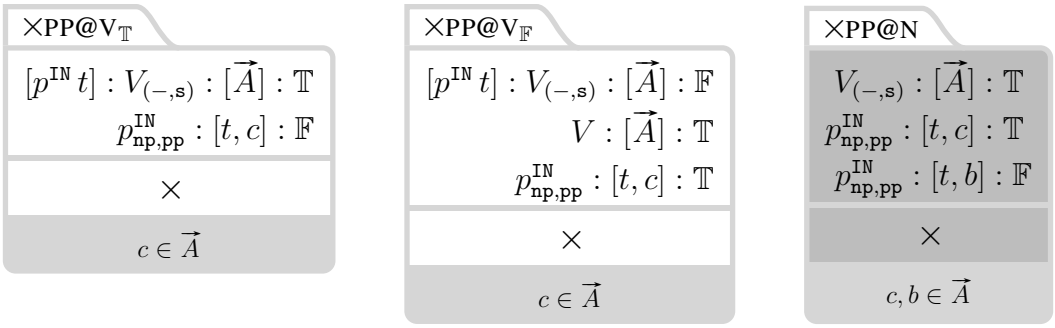
A boy saw a $\big[$criminal$_N$ [with a telescope]$_{N\backslash N}\big]_N$ (8a)
A boy $\big[$[saw a criminal]$_{VP}$ [with a telescope]$_{VP\backslash VP}\big]_{VP}$ (8b)
A criminal was seen by a $\big[$boy$_N$ [with a telescope]$_{N\backslash N}\big]_N$ (9a)
A criminal $\big[$[was seen by a boy]$_{VP}$ [with a telescope]$_{VP\backslash VP}\big]_{VP}$ (9b)

To model the entry as underspecified, we design two closure rules, (×PP@V$_{\mathbb{T}}$) and (×PP@V$_{\mathbb{F}}$). These rules render two LLFs as paraphrases if they differ only in terms of the site of PP attachment where exactly one site of the attachment is a VP. As a result, the rules identify the LLFs of (8a) and (8b) as paraphrases, as well the LLFs of (9a) and (9b).



(a) A→C (b) C→A

Figure 4.5: The tableau proofs for the equivalence of A and C from Figure 4.4



For illustration let us consider the chart in Figure 4.4. The sets of nodes A–D represent the states where a PP is attached to a verb, an object and a subject. The cases are labeled with the corresponding readings of the running examples in (8) and (9).[17] The tableau system with the help of (×PP@V$_{\mathbb{T}}$) and (×PP@V$_{\mathbb{F}}$) is able to prove the equivalence of A and C; see the tableau proofs in Figure 4.5. The equivalence of A and D is proved in the analogous way.

The introduced rules together with (M>) and (M<) prove the equivalence of the LLFs of (8a), (8b), (9a) and (9b) except the equivalence of the LLFs of (8a) and (9a) (see Figure 4.4). The non-transitivity of the *provable* entailment relation, exemplified by the missing entailment link between the C and D states, is caused by treating the entry $[p^{\text{IN}} t_e] : V_{\text{np,vp}} : [c, b]$ as underspecified representation. We could go further and introduce the rule (×PP@N) which identifies the states C and D as paraphrases. Particularly it carries a PP from one argument to another, in contrast to the two previous rules which carry a PP between a verb and its arguments. From the parsing perspective, (×PP@N) is redundant as the parsers rarely attaches a PP to a wrong argument rather than to the correct one. We believe that its adoption more contributes to *unsound* entailments, like the ones between (11) and (12), than the sound ones.

A criminal with a telescope was seen by a boy (11)

A boy with a telescope saw a criminal (12)

John ate sushi with eel (13)

John with eel ate sushi (14)

[17] The analogy between them is established with the following instantiation: $p =$ **with**, $V =$ **see** and b, c, t entities stand for a boy, a criminal and a telescope respectively.

Obviously our approach does not solve the selectional restriction problem associated with PPs (see footnote 10 in § 4.2.1). For example, depending on LLFs, the introduced rules might license entailment relations between (13) and (14). The possible unsoundness introduced by ($\times$PP@V$_{\mathbb{T}}$) and ($\times$PP@V$_{\mathbb{F}}$) is a trade-off between the coverage and precision of reasoning over incorrect PP attachments. It is important to emphasize that the amount of introduced unsoundness is minimal. The presented two rules do not introduce any unsoundness per se as they are closure rules. If a wrong proof is obtained with the help of them, this means that the source derivation trees from the parsers were erroneous.

So now, the natural tableau system, with the help of the new rules, is able to abstract from the site of PP attachments—like the surface forms do—and prove the relations without being hinged on the disambiguated sites of PP attachments. As an example, the system is able to prove the equivalence of (8) and (9), or the entailment of (10) from (8) or (9) regardless the site of PP attachments in the corresponding LLFs. The new rules also enable to classify the textual entailment problems SICK-8145 and SICK-7064 as entailment.

The proposed solution treats PP attachments to a VP as underspecified. This decision not only abstracts from the parsers' mistakes in PP attachments but also from the structurally ambiguous PP attachments. The solution tackles the ambiguity of the site of PP attachment in efficient, modular and simple manner compared to other possible solutions, e.g., employing disambiguated LLFs[18] or modeling disambiguate readings in terms of tableau branches.[19] The closure nature of the rules makes our solution extremely efficient for theorem proving as after application of the rules the branch is closed. Also this nature allows to introduce minimal amount of unsoundness in the tableau system. The modular character of the solution makes it trivial to suppress the solution simply excluding the rules ($\times$PP@V$_{\mathbb{T}}$) and ($\times$PP@V$_{\mathbb{F}}$) from the inventory.

4.2.3 Particles vs prepositions

Particles and prepositions are homographs but differ in their functionality. For instance, "*over*" can be a particle in the *particle-verb construction* (PVC) "*think it over*" and a preposition in the *prepositional verb* (PV) "*jump over it*". The subtle differences between them pose challenges for parsers and for any system that relies on the parsers' output, including our tableau system.[20] In this subsection we introduce rule that help the tableau system to abstract from the problem raised by the particle-preposition distinction. In other words, they contribute to account for the RTE problems, like SICK-1483, regardless of correctly identifying the prepositions and particles.

[18]The approach assumes the generation of several LLFs from a single CCG derivation that are disambiguated readings in terms of the site of PP attachment; Then each combination of LLFs can be checked on logical relations. This approach requires an extra pre-processor for LLFs and has an extremely inefficient proof strategy.

[19]The solution regards certain tableau entries as underspecified, e.g., $[p^{\mathtt{IN}}t_e] : V_{(-,\mathtt{s})} : [\vec{A}] : \mathbb{X}$, and uses the rules that unfold its disambiguated readings. If the disambiguated readings are put on separate tableau branches, then a tableau closes only if any combination of disambiguate readings are in the corresponding entailment relation. This setup would not capture the desired relations over the discussed textual entailments. If the disambiguated readings are placed on the same branch, then they can *interact* with each other which is undesirable. Also this decision ends up with many irrelevant nodes on branches.

[20]The problems usually occur when analyzing the multiword expressions like *phrasal verbs* (i.e. PVCs and PVs). The works that describe these challenges include (Sag et al., 2001), (Baldwin et al., 2009) and (Baldwin and Kim, 2010).

(P$_1$) A deer is $\big[\text{jumping}_{(VP_{ng}/NP)/PR}\ \text{over}_{PR}\big]_{VP_{ng}/NP}$ a cyclone fence

(H$_1$) The deer is $\big[\text{jumping}_{VP_{ng}/PP}\ [\text{over the fence}]_{PP}\big]_{VP_{ng}}$

reC&C, GOLD: ent, SICK-1483

(P$_2$) A deer is $\big[\text{jumping}_{VP_{ng}/NP}\ \text{over}_{VP_{ng}\backslash VP_{ng}}\big]_{VP_{ng}/NP}$ a cyclone fence

(H$_2$) The deer is $\big[\text{jumping}_{VP_{ng}}\ [\text{over the fence}]_{VP_{ng}\backslash VP_{ng}}\big]_{VP_{ng}}$

C&C, GOLD: ent, SICK-1483

Traditionally prepositions are considered to be *transitive*—usually they take an NP as an argument to form a PP. For instance, in the previous subsections we were discussing transitive prepositions. On the other hand, particles do not take any argument and they are themselves the arguments of verbs. Sometimes the concept of the preposition is widened to comprise both particles and traditional (i.e. transitive) prepositions, where particles are regarded as *intransitive* prepositions. We adopt the latter terminology for conveniently referring to both lexical items as prepositions.

In the initial version of CCGbank, the verbs do not subcategorize for particles because of the difficulty to identify PVCs in the Penn Treebank; instead, the particles are treated as adverbial modifiers there (Hockenmaier and Steedman, 2007, 7.2). As a result, the parser trained on this version of CCGbank analyzes particles as of category $VP\backslash VP$, like "*over*" in (P$_2$). After rebanking CCGbank (Honnibal et al., 2010; Constable and Curran, 2009), the new CCG category PR was introduced for particles and they are the arguments of the verbs in PVCs. The new analysis of particles is demonstrated by (P$_1$) where the rebanked C&C wrongly identifies "*jumping over*" as a PVC and analyzes it accordingly.

Due to the particle and transitive preposition distinction and two different treatments of particles in the versions of CCGbank, the CCG parsers can analyze the sequence of a verb, a preposition and an NP at least in four different ways. The examples of these analyses are given above on the sentences of SICK-1483. Our goal is to map all these analyses and the corresponding LLFs to the canonical tableau entry (15), i.e. the entry of the abstract form (16). This reduction automatically leads to the same entries for all those analyses. In this way, the different or wrong CCG derivations of the same surface forms will be reduced to the same entry.

$$\textbf{over}_{\mathtt{np,vp,vp}}\ c_e\ \textbf{jump}_{\mathtt{vp}} : [d_e] : \mathbb{X} \tag{15}$$

$$p^{\mathtt{IN}} C_{\mathtt{np}}\ V_{\vec{\alpha},\mathtt{vp}} : [\vec{E}, D_{\mathtt{np}}] : \mathbb{X} \tag{16}$$

The analyses with the transitive preposition of category PP/NP, like (H$_1$), is reduced to the canonical form (16) with the help of (V@PP) as it was shown in §4.2.2.2. In order to reduce the analyses with the (possibly fake) particles of category PR or $VP\backslash VP$, like (P$_1$) and (P$_2$), to the canonical form, we propose the rules (V@PR) and (PR@V). Both rules deal with the transitive or ditransitive VPs and interpret the particle as a transitive preposition. (V@PR) and (PR@V) are specially designed for the analyses similar to (P$_1$) and (P$_2$), respectively. Notice that (PR@V) makes sure that the lexical term p has a POS tag usually (by mistake) assigned to the particles. The concrete examples of the rule applications are also given below.

V@PR

$[\overrightarrow{M}] : V_{\mathtt{pr},\mathtt{np},\alpha}\ p_{\mathtt{pr}} : [c, \overrightarrow{D}] : \mathbb{X}$

$[\overrightarrow{M}] : p^{\mathtt{IN}}_{\mathtt{np},\alpha,\alpha}\ c\ V_{\alpha} : [\overrightarrow{D}] : \mathbb{X}$

$\alpha = (\mathtt{np}^{+}, \mathtt{s})$

PR@V

$[\overrightarrow{M}] : p^{\mathtt{POS}}_{(\mathtt{np},\alpha),\mathtt{np},\alpha}\ V_{\mathtt{np},\alpha} : [c, \overrightarrow{D}] : \mathbb{X}$

$[\overrightarrow{M}] : p^{\mathtt{IN}}_{\mathtt{np},\alpha,\alpha}\ c\ V_{\alpha} : [\overrightarrow{D}] : \mathbb{X}$

$\alpha = (\mathtt{np}^{+}, \mathtt{s})$ and $\mathtt{POS} \in \{\mathtt{RB}, \mathtt{RP}, \mathtt{TO}, \mathtt{IN}\}$

$$\frac{\textbf{jump}_{\mathtt{pr},\mathtt{np},\mathtt{vp}}\ \textbf{over}_{\mathtt{pr}} : [c, d] : \mathbb{X}}{\textbf{over}^{\mathtt{IN}}_{\mathtt{np},\mathtt{vp},\mathtt{vp}}\ c\ \textbf{jump}_{\mathtt{vp}} : [d] : \mathbb{X}}\ \text{V@PR}^{*} \qquad \frac{\textbf{over}^{\mathtt{RP}}_{\mathtt{vp},\mathtt{vp}}\ \textbf{jump}_{\mathtt{np},\mathtt{vp}} : [c, d] : \mathbb{X}}{\textbf{over}^{\mathtt{IN}}_{\mathtt{np},\mathtt{vp},\mathtt{vp}}\ c\ \textbf{jump}_{\mathtt{vp}} : [d] : \mathbb{X}}\ \text{PR@V}^{*}$$

As we showed on the example of SICK-1483, there can be two different analysis of a particle: one using the category PR and another using $VP\backslash VP$. It is possible that the intransitive verb with a particle is analyzed in both fashion. For example, see the analyses of the sentences of SICK-4117.

A cute panda is [lying$_{VP_{ng}/PR}$ down$_{PR}$]$_{VP_{ng}}$

A cute panda is not [lying$_{VP_{ng}}$ down$_{VP_{ng}\backslash VP_{ng}}$]$_{VP_{ng}}$

reC&C, GOLD: cont, SICK-4117

The case with intransitive verbs is not covered by the above introduced rules. To relate the structurally different terms of "*lying down*", we introduce the additional rule (PR). The rule simply converts an adverbial modifier particle into an argument of the verb. This conversion reduces intransitive PVCs, including the one from SICK-4117, to the canonical form where the particles are treated as arguments. The latter analysis is set as canonical because it represents a more transparent analysis (Constable and Curran, 2009).

PR

$[\overrightarrow{M}] : p^{\mathtt{POS}}_{\mathtt{vp},\mathtt{vp}}\ V_{\mathtt{vp}} : [c] : \mathbb{X}$

$[\overrightarrow{M}] : V_{\mathtt{pr},\mathtt{vp}}\ p_{\mathtt{pr}} : [c] : \mathbb{X}$

$\mathtt{POS} \in \{\mathtt{RB}, \mathtt{RP}, \mathtt{TO}, \mathtt{IN}\}$

$$\frac{\textbf{down}^{\mathtt{RB}}_{\mathtt{vp},\mathtt{vp}}\ \textbf{lie}_{\mathtt{vp}} : [c] : \mathbb{X}}{\textbf{lie}_{\mathtt{pr},\mathtt{vp}}\ \textbf{down}_{\mathtt{pr}} : [c] : \mathbb{X}}\ \text{PR}^{*}$$

The contribution of these rules is demonstrated by the tableau proof for the entailment in SICK-7755 (see Figure 4.11 in Appendix C). Despite two inconsistent analyses by the C&C parser for the same VP, the analyses are still found identical after reducing them to the canonical form with the help of (PR@V).

A man and a woman are walking $\big[$down$_{(VP\backslash VP)/NP}$ [the street of a city]$_{NP}\big]_{VP\backslash VP}$

A man and a woman are $\big[$[walking$_{(VP/NP)/PR}$ down$_{PR}$]$_{VP/NP}$ [a city street]$_{NP}\big]_{VP}$

reC&C, GOLD: ent, SICK-7755

We have illustrated several analyses of PVCs and PVs that are used by the CCG parsers. Since the parsers have hard time with distinguishing these two constructions, they often make mistakes when parsing them. The mistakes can erase logical relations between the surface forms of VPs. We have introduced three rules (V@PR), (PR@V) and (PR) which reduce constructions with verbs and prepositions to the canonical forms. After the reduction, it is easier to detect the erased logical relations.

4.3 Rules for definite noun phrases

The rules for determiners play an important role in the natural tableau system. Remember that determiners have a type-raised type and are the final functions that return the sentential LLFs (i.e. the LLFs of type `s`). Due to this reason, the rules for determiners are the very first rules that start unfolding the semantics of the sentential LLFs. The tableau rules for logical determiners "*every*" and "*some*" are already given by Muskens (2010), where the rule for "*some*" also serves as a rule for "*a*" or "*an*". In the section, we model semantics of a singular definite NPs (i.e. *definite descriptions*) of the form "*the* N", where N is a bare singular noun. Modeling semantics of the definite NP is itself a problematic issue. First, we present two mainstream theories of definite descriptions and show that their straightforward integration in the tableau system gives wrong predictions for textual entailments. Then, instead of giving a refined theory for definite descriptions, which is beyond the scope of the thesis, we propose several solutions that are simple, efficient for theorem proving and adhere to natural reasoning.

4.3.1 Two theories of definite descriptions

There are two main approaches to model semantics of the singular definite NPs in formal logic. The first approach is due to Frege which later was defended by Strawson (1950), and the second one is proposed by Russell (1905). We briefly present each approach and show the problems of integrating them in the natural tableau system.

4.3.1.1 Frege's analysis

According to Frege and Strawson (1950), the meaning of "*the* N" is defined if and only if there exists only one entity in the extension of the noun phrase N. In other words, the meaning of "*the* N" presupposes the *existence and uniqueness* of the entity satisfying N. From the LLFs' perspective, this means that the denotation of $\mathbf{the}_{\mathtt{n},\mathtt{vp},\mathtt{s}}$ is the partial function that is defined on singleton sets. The partiality of $\mathbf{the}_{\mathtt{n},\mathtt{vp},\mathtt{s}}$ raises an issue so-called *gaps in the truth values*. In particular, a proposition involving a definite description does not have a truth value in the situation where the existence and uniqueness presupposition fails. For example, the truth value of the sentence in (17) is not defined when considering a situation where (18) holds (and there is no context that singles out a particular boy). On the other hand, if truth value of (17) is defined in a situation, then a unique boy is assumed which contradicts the semantics of (18).

The boy is running (17)

Two boys are running (18)

The freckled boy is running (19)

The gaps in the truth values violate the *law of excluded middle*, which means that sentences like "*the boy is sleeping or the boy is not sleeping*" are not necessary truths (e.g., consider the case where (18) holds). While having the truth gaps, it seems impossible to model the entailment relation with a two-signed tableau. For example, the situation where P is true and Q is false, i.e. $\{P : \mathbb{T},\ Q : \mathbb{F}\}$, is not the only counterexample for "P *entails* Q" (formally, $P \models Q$). The situation where P is true and Q's truth value is undefined

is also the counterexample for the entailment. So, the closure of the tableau initiated with $\{P : \mathbb{T}, Q : \mathbb{F}\}$ is not sufficient to prove $P \models Q$. Additionally the tableau built over $\{P : \mathbb{T}, Q : \mathbb{U}\}$ needs to be closed to make sure that there are no counterexamples, where $\mathbb{U}$ is assumed to stand for the *undefined* truth value. At this stage, introduction of additional truth values in the natural tableau system seems to us as an unnecessary complication of the system.

One option for modeling entailment with two signs is to understand validity in terms of *Strawson's validity* (von Fintel, 1999). This change brings a new concept of entailment, let it be *P-entailment*, which is defined as follows: S *P-entails* T if and only if S and R (classically) entail T, where R is a collection of all presuppositions of S and T. Let us neglect the obscure concept of presupposition in the tableau system for a moment. It is easy to check that S-entailment makes "*the*" downward monotone in its first argument, e.g., (17) S-entails (19), that is unwanted. Moreover, if S contradicts the presupposition of T, then S P-entails T. For instance, "*there are no boys*" P-entails (17). Due to these reasons we consider the P-entailment as impractical for modeling the natural language entailment (at least from the RTE perspectives).

4.3.1.2 Russell's analysis

In contrast to the former approach, Russell (1905) proposes the semantics for *the N* which avoids the gaps in truth values. In particular, semantics of a sentence of the form "*the N V*" is defined as a conjunction of three statements: there exists an entity that is N, this entity is the only N, and every N is V. The lexical semantics of "*the*" is concisely expressed in first-order logic as:

$$[\![the]\!] = \lambda SP.\big(\exists x\big(Sx \wedge \forall y(Sy \rightarrow x = y) \wedge Px\big)\big)$$

where S and P are variables for unary predicates standing for a noun subject and an intransitive verb predicate, respectively.

Russell suggests incorporating the existence and uniqueness presupposition into the semantics of the definite description. This makes a sentence of the form "*the N V*" false when the presupposition is not met; hence the truth value of the sentence is always defined. Unfortunately, the approach has problems as it does not license certain entailments that seem *naturally valid*. These kind of entailments are those, similar to SICK-9604, where the conclusion mainly has definite articles while the premise is using indefinite ones. For example, according to the Russellian analysis SICK-9604 is neutral as the premise does not assert the uniqueness of "*dog*" and "*ball*" while the conclusion does.[21] Another type of textual entailments that are wrongly classified by the Russellian analysis reveals the upward monotonicity of "*the*" in the first argument position (see SICK-3039). The textual entailments similar to SICK-3039 are classified as neutral by the running approach—the uniqueness presupposition of the conclusion is stronger compared to the premise.[22] Though Russell's approach is formally well-defined and avoids the gaps in

[21] The counterexample for the entailment in SICK-9604 would be the situation where there are two black and white dogs and one of them is jumping for some unique ball. The situation makes the premise true but the conclusion false.

[22] For instance, the counterexample for the entailment in SICK-3039 is the situation where the unique

truth values, its uniqueness presupposition is not restricted to some relevant context. As a result the approach fails to capture certain naturally occurring entailments.

A black and white dog is jumping for a ball

The black and white dog is jumping for the ball

GOLD: ent, SICK-9604

The puppy is playing with a plastic container

The dog is playing with a plastic container

GOLD: ent, SICK-3039

To sum up, Frege's approach comes with truth gaps that occur when the presuppositions carried by the definite descriptions are not satisfied. At the same time, modeling the gaps or assuming the presuppositions in the tableau system carries much of complication or unsound entailments, respectively. On the other hand, Russell's analysis pushes presuppositions in the semantics of the definite descriptions; this makes the approach too harsh and inadequate to some extent for natural reasoning. Both approaches fall short to account for the entailment relation found in the RTE datasets.

4.3.2 Two options for modeling definite NPs

In the previous subsection, it was shown that neither Frege's nor Russell's account for the definite descriptions, i.e. phrases like *the N*, are suitable for the natural tableau and natural reasoning. In this subsection, we suggest two simple, efficient and comparably adequate analyses of definite descriptions in the tableau system.

4.3.2.1 Definite NPs as indefinite NPs

The first proposal is to treat "*the*" as the indefinite determiner "*a*". This approach is too simple and is automatically accommodated in the system. Despite its simplicity, the approach works well concerning the entailment problems similar to SICK-3039 and SICK-9604. For these reasons, identifying "*the*" with "*a*" is a default approach for many RTE systems in the wide-coverage semantic analysis.

There is one issue with this approach. It fails to render the definite NPs as referring expressions. For example, if "*the*" is treated as "*a*", then it will not be possible to prove the entailment relation in FraCaS-99: "*the demonstration*" in (P2) will not refer to the entity that is introduced by "*the demonstration*" in (P1), hence the link between "*Smith*" and "*the system's performance*" will not be captured. The same issue prevents the tableau system to capture the contradiction relation in SICK-1869 and SICK-1480.[23]

puppy is playing with some plastic container and some senior dog is sleeping nearby. There are plenty of textual entailments similar to SICK-3039 in the SICK datasets (Marelli et al., 2014b), where all of them are judged by human annotators as entailment.

[23] In case of SICK-1480, if the negation takes a scope over the entire sentence, then it is possible to prove the contradiction relation. But this rather seems an ad hoc remedy.

P1: Clients at the demonstration were all impressed by the system's performance
P2: Smith was a client at the demonstration

Smith was impressed by the system's performance

GOLD: `ent`, FraCaS-99

A parrot is speaking

The parrot is silent in front of the microphone

GOLD: `cont`, SICK-1869

A deer is jumping over a cyclone fence

The deer is not jumping over the fence

GOLD: `cont`, SICK-1480

Despite the mentioned drawback, the treatment of the definite NPs as indefinites gives decent predictions. Moreover, its integration in the natural tableau is for free as it requires no new rules. Due to these reasons, this approach can be considered as a baseline solution for modeling the definite NPs in the natural tableau. Another approach, which is presented next, represents a refined version of the current one.

4.3.2.2 Definite NPs as referring expressions

We present the approach that borrows the idea of treating the definite NPs as indefinites from the previous one but also tries to interpret them as referring expressions. While doing so, we aim to come up with the efficient treatment of definite NPs in the tableau system as they are frequently occurring in open-domain text.

In order to interpret the definite NP as a referring expression, one should design rules that emulate the search for the referent of the NP. The definite NPs are assumed to have a unique referent in the context. We have seen in § 4.3.1.1 that presupposing the existence and uniqueness of the referent makes "*the*" as a downward monotone operator, which itself leads to the unsound entailments (e.g., (17) entails (19)). Additionally, modeling the uniqueness condition requires the introduction of the equality operator and the rules associated with it, which all together is usually challenging to be handled efficiently in theorem proving. Taking into account the previous approach (see § 4.3.2.1), which is decent and comes for free, our aim is to suggest another alternative approach that is still cheap, efficient for theorem proving and better accounts for natural reasoning.

In a new approach, we extend ($\forall_{\mathbb{T}}$) in such a way that the rule also applies to the definite NPs in a true context (i.e. with $\mathbb{T}$). In this way, the definite NPs in a true context act as *restricted universal quantifiers*. This interpretation contributes to treat the definite NPs as referring expressions. According to ($\forall_{\mathbb{T}}$), any entity that falls in the description of N is taken as a referent in the right branch while the left branch immediately closes. This approach follows the Russellian analysis in the usage of the universal quantifier in the semantics of "*the*", but it does this only in a true context. If no entity satisfies the description, then the definite article is treated as indefinite one by the ($\exists_{\mathbb{T}}$). In a false context we analyze the definite descriptions as indefinite via ($\exists_{\mathbb{F}}$). This prevents them from introducing a fresh entity in a false context.

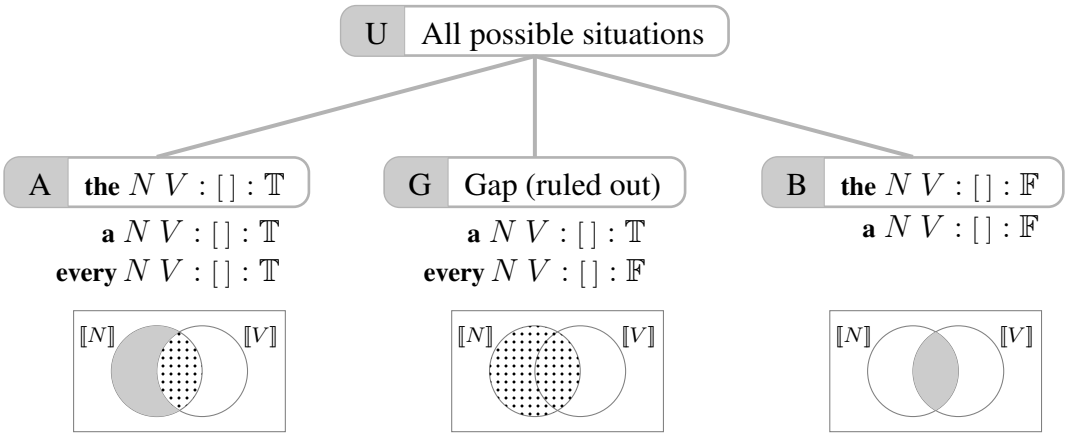
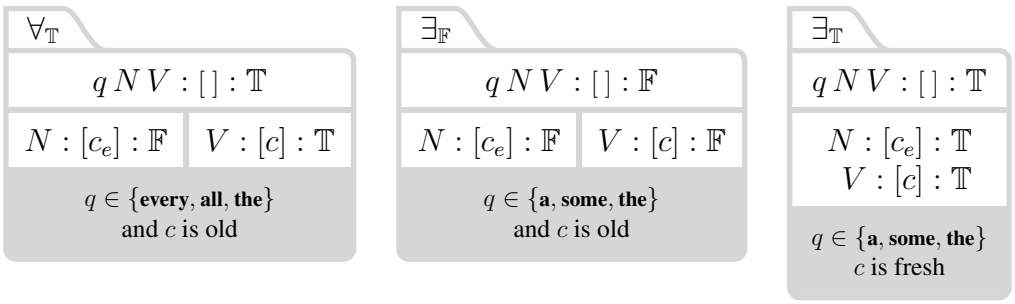


Figure 4.6: The partition of possible situations according to the truth values of **the** NV. Each partition is accompanied with a Venn diagram, where dotted, gray and white areas are non-empty, empty and arbitrary, respectively. The situations where the truth value is not defined represents a case for the truth gap. These situations are ruled out by the presupposition associated with **the** NV.



Let us see whether the natural tableau, with such treatment of definite NPs, accounts for the above mentioned problematic entailments. The definite article "*the*" is not treated as downward monotone because if the LLF of the form **the** NV is marked with the false sign, it does not assert or presuppose the existence of the referent. Therefore, (17) does not entail (19). With the help of ($\exists_{\mathbb{T}}$) and ($\exists_{\mathbb{F}}$), "*the*" is analyzed as the indefinite article, which allows the proofs for the problems like SICK-3039 and SICK-9604. The definite NPs marked with the true sign are treated as referring expressions via ($\forall_{\mathbb{T}}$), as it was already discussed in the previous paragraph. As a result, "*the demonstration*" in the premises of FraCaS-99 co-refer to the same entity—one of the definite NPs introduces the entity via ($\exists_{\mathbb{T}}$) and another one will co-refer to it via ($\forall_{\mathbb{T}}$). The same scenario applies to the co-references in SICK-1480 and SICK-1869. Notice that the assertions are also made by ($\exists_{\mathbb{F}}$) on any entity that satisfies the description.

In order to see what semantics this approach assigns to the LLFs of the form **the** NV, we give a partition of all possible situations with respect to the truth value of such LLFs (see Figure 4.6). **the** NV being false amounts to the semantics of the false **a** NV because both types of LLFs are only decomposed with the same ($\exists_{\mathbb{F}}$) rule in a false context. The set of situations that falsify **the** NV (i.e. **a** NV) is denoted by **B** and accompanied with the corresponding Venn diagram. Let **A** be the set of situations where **the** NV holds. We know that ($\exists_{\mathbb{T}}$) and ($\forall_{\mathbb{T}}$) are applicable to **the** NV : [ ] : $\mathbb{T}$ and decomposes it in the same way as **a** NV : [ ] : $\mathbb{T}$ and **every** NV : [ ] : $\mathbb{T}$. So, the situations in **A** are exactly those ones

that make **a** NV and **every** NV together true.

Apparently the sets **A** and **B** do not cover all possible situations. The uncovered set is denoted by **G** and represents a collection of those situations where **a** NV holds and **every** NV does not. In other words, the gap in truth values of **the** NV occurs if and only if there are at least two entities that satisfy the description N and are distinguishable via the predicate V. The truth gap means that the LLFs of the form **the** NV carry some presuppositions—ruling out the cases of the gap. Compared to the Fregean analysis, our approach carries less presuppositions and hence rules out less situations—put differently, the truth gap here is *smaller* than in case of the Fregean analysis. If we look at the presupposition of the current approach closely it seems quite natural. For example, the truth value of "*the boy is running*" in the situation where one boy is running and another is not is ambiguous and depends on the referent. On the other hand, it seems unsurprising to evaluate "*the dog is running*" as false in the situations where there are no dogs running. Moreover, the assertion "*the boy is running*" in the situations where there are some boys who are all running can still survive.

We have outlined the Fregean and Russellian analyses of definite descriptions and showed how they fall short to account for natural reasoning. Two simple and relatively efficient solutions were proposed for modeling the definite NPs in the tableau system. None of them require introduction of new rules. While the first one is slightly efficient than the second one, the latter gives more adequate analysis of the definite descriptions, i.e. it models them as referring expressions, that goes well with natural reasoning. Both solutions will be tested against the textual entailment problems in §6.2.2.

4.4 Closure rules

A closure rule is a rule that identifies inconsistency in a tableau branch, and as a result, introduces the closure sign $\times$ that closes the branch. We have already discussed three general closure rules ($\times\sqsubseteq$), ($\times|$) and ($\times\smile$) in Chapter 2 and another two, but specific, closure rules ($\times$PP@V$_{\mathbb{T}}$) and ($\times$PP@V$_{\mathbb{F}}$) in §4.2.2.3. In this section additional closure rules are introduced. According to their nature, the rules are specially design for certain linguistic phenomenon, like subcategorization, or constructions involving expletives, light verbs and compound nouns. Some of those rules represent efficient but incomplete rules. Incompleteness is due to the incomplete lists of compound nouns or light verb constructions. Moreover, it is safe to use closure rules for modeling constrictions that are restricted to a specific list of words. In this way, it is guaranteed that the tableau rules will not infer any meaningless or unsound entries on tableau branches.

4.4.1 The rule for expletive *there*

There is significant amount of textual entailment problems in RTE datasets that involve the expletive expression "*there*". For example, nearly 6% and 3% of the sentences in SICK and FraCaS, respectively, contain the expletive "*there*". From the coverage viewpoint, it is important to model the expletive constructions, like (20), in the natural tableau.

In the sentences like (20), "*there*" is considered as a semantically vacuous element that fills the subject position and has no corresponding realization in the logical form (LF) level

(Chomsky, 1986). The LF of (20) is derived by substituting "*there*" with "*some animal*", which is bound by the expletive; this fact is encoded with the subscripts. Hence (20) and (21) have the same semantics in the end.

There$_1$ is [some animal]$_1$ moving (20)

$\mathbf{some}_{\mathrm{n,vp,s}}\ (\mathbf{that}_{\mathrm{vp,n,n}}\ \mathbf{move}_{\mathrm{vp}}\ \mathbf{animal}_{\mathrm{n}})(\lambda x.\ \mathbf{be}_{\mathrm{np,vp}}\ x_{\mathrm{np}}\ \mathbf{there}_{\mathrm{np}_{thr}})$ (20a)

Some animal is moving (21)

$\mathbf{some}_{\mathrm{n,vp,s}}\ \mathbf{animal}_{\mathrm{n}}\ \mathbf{move}_{\mathrm{vp}}$ (21a)

There are several persons in the room (22)

There are several companies (23)

On the other hand, the expletives are presented in the LLFs that we obtain from the CCG derivations, e.g., see the LLF in (20a).[24] It is possible to carry out the similar substitution on the LLFs with the help of the (THERE) rule and obtain more semantically transparent LLFs as a result. Unfortunately, the rule cannot process the LLFs of the sentences (22) and (23) in the same manner since the common nouns of those LLFs have different structures than $r\ V_{\mathrm{vp}} N_{\mathrm{n}}$.

THERE

$$q_{\mathrm{n,vp,s}}\ (r\ V_{\mathrm{vp}}\ N_{\mathrm{n}})(\lambda x.\ \mathbf{be}_{\mathrm{np,vp}}\ x\ \mathbf{there}_{\mathrm{np}_{thr}}) : [\,] : \mathbb{X}$$

$$q_{\mathrm{n,vp,s}}\ N\ V : [\,] : \mathbb{X}$$

×THR

$$\mathbf{be}_{\mathrm{np,vp}} : [C, D] : \mathbb{F}$$

$$\times$$

$$\mathbf{there}_{\mathrm{np}_{thr}} \in \{C, D\}$$

Instead of repairing (THERE) to accommodate (22) and (23) cases, we give different rules that are more general. The idea that makes those rules general is simple and suggests interpreting the expletives with the copula as the universal predicate. In other words, the terms $\lambda x.\ \mathbf{be}\ x\ \mathbf{there}$ and $\mathbf{be\ there}$ are true for any argument. We introduce the rule (×THR) that models this universality feature. When $D = \mathbf{there}$, the rule renders the situation impossible where, for example, there is an individual which is a dog and "*there is some dog*" does not hold in the situation. In case of $C = \mathbf{there}$, the rule does the similar job but for the more rare paraphrase "*Some dog is there*". Notice that the modifier list in the rule is required to be empty.

The tableau proof in Figure 4.7 makes use of (×THR) in order to prove the entailment relation in PROB-8.[25] One could notice that the rule (×THR) makes the sentences, like "*there is John*", always true, i.e. tautology. This fact does not represents a big issue if a weak reading of the sentence "*there is John*" is considered, where "*there is*" is understood as "*exists*" rather than presence at a particular location.

Some dog is running

There is some animal moving

PROB-8

[24] Notice the insertion of the relative pronoun in the LLF. This is the technique adopted in Chapter Y for explaining the type-changing rule $\mathrm{vp} \mapsto (\mathrm{n}, \mathrm{n})$ for restrictive participial phrases. Moreover, the expletives are analyzed as NPs by the CCG parsers, but their category comes with the feature thr.

[25] Another proof that makes use of (×THR) is presented in Figure 4.10 (Appendix C). The proof classifies 1417 as contradiction.

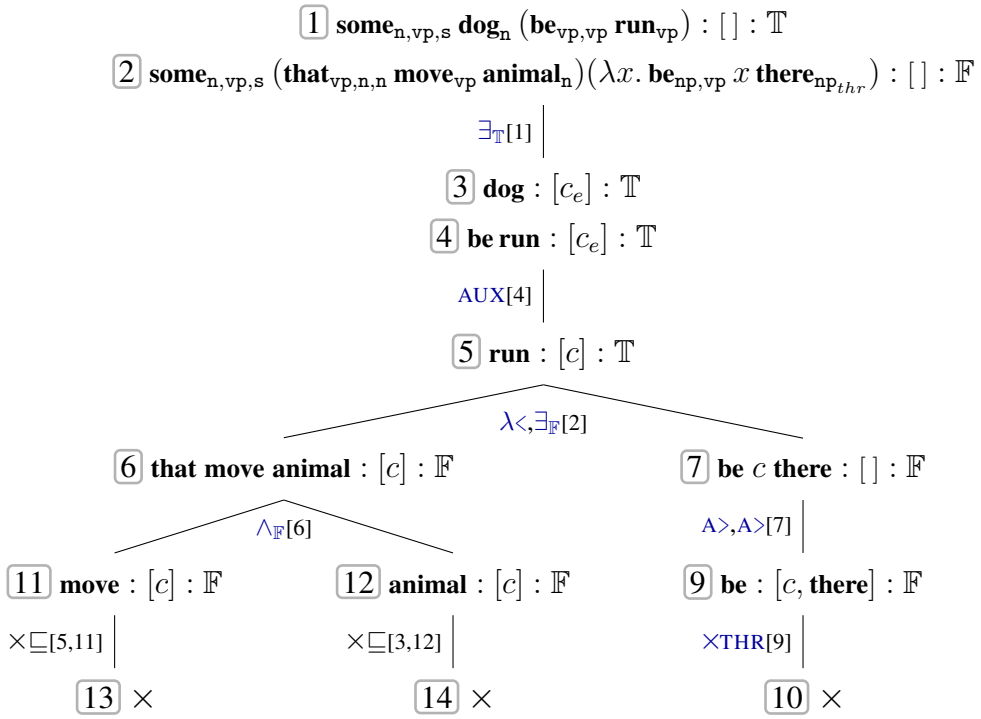


Figure 4.7: The closed tableau proves PROB-8 as entailment. The rightmost branch is closed by the (×THR) rule while the rest of the branches are closed by (×⊑). We use a sequence of rules in rule applications in order to omit irrelevant intermediate nodes. For instance, λ<,$\exists_{\mathbb{F}}$[2] assumes that 6 is obtained by $\exists_{\mathbb{F}}$ and 7 by λ< and $\exists_{\mathbb{F}}$ from 2.

We have presented the rule (×THR) which analyzes the expletive-copula pair "*there is*" as the universal predicate of type vp. The rule treats various construction, including (20), (22) and (23), in a uniform way.

4.4.2 Verb subcategorization

There are several types of verbs that subcategorize for optional arguments. For example, the verbs, like "*eat*" and "*read*", can take an optional object, see (24) and (25). Also certain passive constructions can be seen as a case of subcategorization: consider (26) and (27). In order to prove the entailment of (24) from (25) or the contradiction between (26) and (27), we could use event semantics and (EV$_{\mathbb{T}}$) from §2.3.2. As we have already noted there, (EV$_{\mathbb{T}}$) requires information about thematic roles of a verb and its counterpart rule for $\mathbb{F}$ represents an inefficient γ-rule.

John ate yesterday (24)

John ate [khinkali]$_{NP}$ yesterday (25)

Mary sold a car to John (26)

A car was not sold to John (27)

In order to efficiently capture semantic relations licensed by verb subcategorization, we introduce two rules (×VSC$_1$) and (×VSC$_2$). The rules model subcategorization for active

and passive verbs respectively. Notice that the rules do not introduce any event entity or a new tableau entry; they simply identify contradictions, and this makes them efficient.

$\times$VSC$_1$

$$[\overrightarrow{M}^{\uparrow}] : U_{\vec{\alpha},\vec{\beta},\mathtt{s}_i} : [\vec{C}_{\vec{\alpha}}, \vec{D}_{\vec{\beta}}] : \mathbb{T}$$
$$[\overrightarrow{M}^{\uparrow}] : V_{\vec{\beta},\mathtt{s}_i} : [\vec{D}_{\vec{\alpha}}] : \mathbb{F}$$

$\times$

$\vec{C}$ and $\vec{D}$ are nonempty.
$\vec{\alpha}, \vec{\beta}$ involve only pp and np.
If U and V are lexical,
$U \leq V$; otherwise $U = V$

$\times$VSC$_2$

$$[\overrightarrow{M}^{\uparrow}] : U_{\vec{\alpha},\vec{\beta},\mathtt{vp}_i} : [\vec{C}_{\vec{\alpha}}, \vec{D}_{\vec{\beta}}, b] : \mathbb{T}$$
$$[\overrightarrow{M}^{\uparrow}] : V_{\vec{\beta},\mathtt{s}_{pss}} : [\vec{D}_{\vec{\beta}}] : \mathbb{F}$$

$\times$

$\vec{D}$ is nonempty. $i \neq pss$.
$\vec{\alpha}, \vec{\beta}$ involve only pp and np.
If U and V are lexical,
$U \leq V$; otherwise $U = V$

The first instance of the rule identifies the tableau entries like ones in (28) as contradiction; the second instance contrasts passive and non-passive VPs to each other and as a result finds the entries similar to those in (29) contradictory. Notice that the rules can apply to the nodes with nonempty modifier lists, e.g., (28). The rule ($\times$VSC$_1$) in action is demonstrated by the tableau proof of 1417 in Figure 4.10 (Appendix C).[26]

$$[\textbf{yesterday}_{\mathtt{vp},\mathtt{vp}}] : \textbf{swallow}_{\mathtt{np},\mathtt{vp}_{dcl}} : [k, j] : \mathbb{T} \quad (28)$$
$$[\textbf{yesterday}_{\mathtt{vp},\mathtt{vp}}] : \textbf{ate}_{\mathtt{vp}_{dcl}} : [j] : \mathbb{F}$$

$$\textbf{sell}_{\mathtt{pp},\mathtt{np},\mathtt{vp}_{dcl}} : [\textbf{to}\ j, c, m] : \mathbb{T} \quad (29)$$
$$\textbf{sell}_{\mathtt{pp},\mathtt{vp}_{pss}} : [\textbf{to}\ j, c] : \mathbb{F}$$

VSC

$$v_{\mathtt{np},\mathtt{vp}_i} : [C, D] : \mathbb{T}$$

$$v_{\mathtt{vp}_i} : [D] : \mathbb{T}$$
$$v_{\mathtt{vp}_{pss}} : [C] : \mathbb{T}$$

$i \neq pss$

Men are sawing logs

There are no men sawing

GOLD: ent, SICK-1417

Definitely this is not the only way of modeling verb subcategorization in the tableau system. Another similar solution is to design non-closure rules that *shrink* the argument structure of the verb, e.g., if they find the entry $\textbf{eat}_{\mathtt{np},\mathtt{vp}} : [k, j] : \mathbb{T}$ they will introduce $\textbf{eat}_{\mathtt{vp}} : [j] : \mathbb{T}$ and $\textbf{eat}_{\mathtt{vp}_{pss}} : [k] : \mathbb{T}$. This solution is represented by the rule (VSC). This approach, with a non-closure rule, is general but it is inefficient compared to our previous solution. Moreover, such a non-closure rule can introduce several new entries from the first LLF of (29), possibly including unsound ones like $\textbf{sell}_{\mathtt{vp}} : [m] : \mathbb{T}$. In contrast, our suggested solution does not introduce any new entries, especially the unsound ones.

The suggested solution employs the closure rules ($\times$VSC$_1$) and ($\times$VSC$_2$), and compared to the alternative solution with event semantics, it is significantly more efficient. The solution does not introduce any new tableau entries and employs only those that are derived from the CCG derivations. Moreover, the proposed rules carry out shallow

[26] Careful observation on the node 6 of the tableau shows that the LLF for the premise is analyzed according to the *distributive* reading rather than the *cumulative* one.

reasoning over syntactic terms, and hence the solution is faithful to the project of natural logic. Due to its efficiency, the solution is adopted in a computational model of the natural tableau prover (Chapter 5).

4.4.3 Open compound nouns

A compound noun is formed from more than one stems and acts as a semantic unit.[27] Compound nouns are quite frequent in the wide-coverage text and the RTE datasets. Accordingly, in this subsection, we introduce a single rule that captures the equivalence relation between open compound nouns and noun phrases that are formed from the similar words, for example, "*beer bottle*" is "*bottle for beer*" and vice versa.

Let us consider the entailment problems SICK-3275 and SICK-7755 from the SICK dataset. Up to now there is no rule or transformation of LLFs that captures the equivalence of the noun phrases "*waves of the ocean*" and "*ocean waves*" or "*street of a city*" and "*city street*". Hence, the tableau system fails to classify correctly these problems and similar ones.

A child is running in and out of the waves of the ocean

A child is running in and out of the ocean waves

GOLD: ent, SICK-3275

A man and a woman are walking down the street of a city

A man and a woman are walking down a city street

GOLD: ent, SICK-7755

We introduce a single closure rule (×CPN) that enables the tableau system to capture the equivalence relation between a two-word compound noun (e.g., "*ocean wave*") and its paraphrase with a preposition (e.g., "*wave of the ocean*"). Each instantiation of the truth variable $\mathbb{X}$ corresponds to one of the entailment relations of the equivalence. For instance, the contradiction in case of $\mathbb{X} = \mathbb{T}$ models the entailment of "*wave of the ocean*" from "*ocean wave*" while the entailment in the opposite direction is captured by $\mathbb{X} = \mathbb{F}$.

×CPN

$$\begin{array}{r} N_{\mathtt{n}} : [d] : \mathbb{T} \\ [\overrightarrow{M}] : H_{\mathtt{pp,n}}(p^{\mathtt{IN}}_{\mathtt{np,pp}}\ d) : [c] : \overline{\mathbb{X}} \\ [\overrightarrow{M}] : A_{\mathtt{n,n}}\ H_{\mathtt{n}} : [c] : \mathbb{X} \\ \hline \times \\ N \approx A \text{ or } N \approx_d A \end{array}$$

$$\frac{\begin{array}{r} \textbf{protection}_{\mathtt{n}} : [d_e] : \mathbb{T} \\ \textbf{gear}_{\mathtt{pp,n}}(\textbf{for}_{\mathtt{np,pp}}\ d_e) : [c_e] : \mathbb{F} \\ \textbf{protective}_{\mathtt{n,n}}\ \textbf{gear}_{\mathtt{n}} : [c_e] : \mathbb{T} \end{array}}{\times}(\times\text{CPN}^{*})$$

For wider-coverage we allow relaxed constraints on the post-nominal term $N_{\mathtt{n}}$ and the pre-nominal term $A_{\mathtt{n,n}}$. Apart from $N_{\mathtt{n}}$ and $A_{\mathtt{n,n}}$ having the same string representations,

[27] The following words are compound nouns: "*football*", "*whiteboard*", "*beer bottle*", "*father-in-law*", etc. Depending on their form they are further classified as *solid* (e.g., the first two compounds), *open* (e.g., the third compound) and *hyphenated* (e.g., the last compound).

denoted as $N \approx A$ (e.g., $\textbf{ocean}_{\mathtt{n}} \approx \textbf{ocean}_{\mathtt{n,n}}$), the terms are also allowed to be derivations of the same stem, written as $N \approx_d A$.[28] This helps to identify the synonyms like "*protective gear*" and "*gear for protection*" as $\textbf{protection}_{\mathtt{n}} \approx_d \textbf{protective}_{\mathtt{n,n}}$, see ($\times$CPN*). In the same spirit, to increase the coverage of the rule, we do not specify the prepositional term $p^{\mathtt{IN}}_{\mathtt{np,pp}}$. The tableau proof in Figure 4.11 (Appendix C) demonstrates the usage of ($\times$CPN).

From the perspective of theorem proving, our suggested solution models the synonymy relation in the efficient way. This is explained by two properties of the rule ($\times$CPN). The first property is the closure nature of the rule—after applying the rule the entire branch closes. The second property is the *subterm property*. A rule has this property if its consequents contain only subterms of the antecedent terms.[29] The subterm property is automatically obtained if a rule is closed. We emphasize the subterm property of ($\times$CPN) because it is a very important feature for the efficiency when modeling the *open* (i.e. varying) class of construction, like compound nouns. Consider the solution with the rules (CPN_1) and (CPN_2), which do not have the subterm property.[30] Informally speaking (CPN_1) can introduce "*ocean wave*" from "*wave of the ocean*" and (CPN_1) in the other way around. These rules can easily produce garbage on the branches. For example, (CPN_1) introduces the term for the non-existing compound "*day song*" if it is applied to the corresponding terms modeling "*song of a day*". There is also an additional problem with guessing the preposition $p_{\mathtt{np,pp}}$ in a paraphrase. When (CPN_2) introduces a new entry an adequate preposition needs to be chosen: while "*ocean wave*" is paraphrased as "*wave of the ocean*", "*washing machine*" is paraphrased as "*machine for washing*". In some cases, e.g. "*cell phone*", there is no corresponding preposition or paraphrase that (CPN_2) can introduce.

CPN_1

$$\frac{n_{\mathtt{n}} : [d] : \mathbb{T} \qquad h_{\mathtt{pp,n}}(p^{\mathtt{IN}}_{\mathtt{np,pp}}\ d) : [c] : \mathbb{T}}{n_{\mathtt{n,n}}\ h_{\mathtt{n}} : [c] : \mathbb{T}}$$

CPN_2

$$\frac{n^{\mathtt{NN}}_{\mathtt{n,n}}\ h_{\mathtt{n}} : [c] : \mathbb{T} \qquad n_{\mathtt{n}} : [d] : \mathbb{T} \qquad h_{\mathtt{pp,n}}(p^{\mathtt{IN}}_{\mathtt{np,pp}}\ d) : [c] : \mathbb{T}}{d \text{ is fresh}}$$

The rule ($\times$CPN) represents a simple and efficient solution for detecting synonymous paraphrases of open compound nouns. It contrasts the terms obtained from the linguistic representations and avoids introduction of non-existing compounds or paraphrases. The specificity of the antecedents of the rule (i.e. three entries with certain structures) and the relaxed constraints on the terms (i.e. the unspecified preposition and the loose matching with $\approx$ and $\approx_d$) can be seen as a balance between precision/soundness and coverage/completeness.

[28] The string equality modulo derivation $\approx_d$ is defined only for lexical terms while the string equality $\approx$ is defined for any λ-terms, including complex terms too.

[29] This property is usually called as the *subformula property* but due to the shift in terminology we refer it as the subterm property.

[30] These rules are simplified for the demonstration purpose and do not represent the alternative solution. We also restrict the meta-variables to lexical terms since changing the type of a term is trivial on lexical level.

4.4.4 Light verb constructions

A verb is called *light* if it has little semantic content. The light verbs usually form a VP in combination with a noun derived from a verb. The examples of light verb constructions are: "*do a dance*", "*give a presentation*" and "*have a sleep*", where "*dance*", "*present*" and "*sleep*" are their verb synonyms, respectively. A textual entailment problem involving a light verb construction is presented in SICK-253.

A hiker is on top of the mountain and is doing a joyful dance
A hiker is on top of the mountain and is dancing
GOLD: ent, SICK-253

In order to capture the semantic equivalence between the light verb constructions and their verb synonyms, we introduce the closure rule (×LVC) in the rule inventory.

×LVC

$$\frac{[\overrightarrow{M}] : l_{\overrightarrow{\alpha},\mathtt{vp}} : [c, \overrightarrow{D}] : \mathbb{X} \qquad u_{\mathtt{n}} : [c] : \mathbb{T} \qquad [\overrightarrow{M}] : v_{\overrightarrow{\alpha},\mathtt{s}} : [D] : \overline{\mathbb{X}}}{\times}$$

$l \in \{\textbf{do}, \textbf{get}, \textbf{give}, \textbf{have}, \textbf{make}, \textbf{take}\}$, $\overrightarrow{\alpha}$ is formed by np and pp, and $u \approx_d v$

$$\frac{\textbf{do}_{\mathtt{np,vp}} : [d_e, h_e] : \mathbb{T} \qquad \textbf{dance}_{\mathtt{n}} : [d_e] : \mathbb{T} \qquad \textbf{dance}_{\mathtt{vp}} : [h_e] : \mathbb{F}}{\times}\,(\times\text{LVC}^*)$$

The rule is efficient as it is a closure rule and has the subterm property—introduces only subterms found in the antecedents. Because we do not employ a predefined list of light verb constructions, the lexical terms are unspecified.[31] In this way we increase the coverage of the rule. Modeling the phenomena with (×LVC) guarantees that no new terms are introduced in a tableau: for example, introducing the LLF of the non-existing verb for "*homework*" from the entries modeling "*do the homework*". The rule contributes to prove the entailment in SICK-253. In the tableau proof for the entailment relation, the entries to which (×LVC) applies are given in (×LVC*).

We suggest the single rule (×LVC) that models the open class of light verb constructions in the same vein as the compound nouns were modeled by (×CPN) in §4.4.3.

4.5 Rules for the copula *be*

From the semantic point of view, the copula "*be*" is considered as a semantically vacuous word in predicative sentences like (30) and (31), and it is usually omitted in logical forms (Heim and Kratzer, 1998, Ch. 4). Instead of omitting the copula directly in the LLFs, we keep it there (e.g., see the corresponding LLFs) and discard it later with the help of the rules during the tableau proof construction.

Xavi is short (30)

[31] Even $u_{\mathtt{n}}$ and $v_{\overrightarrow{\alpha}}$ are not required to be string equal to each other; they can be forms derived from the same stem, like in "*give a presentation*" and "*present*".

be short Xavi (30a)

Xavi is on the pitch (31)

the pitch $\big(\lambda x.$**be** $\big($**on** $x\big)$ **Xavi**$\big)$ (31a)

The rule (ID) treats the copula as the identity function. When $\textbf{be}_\alpha$ takes a predicative adjective as its argument, like in (30a), $\alpha = (\texttt{vp}, \texttt{vp}_{adj})$ since the predicative adjectives get $\texttt{vp}_{adj}$ category by the CCG parsers. α matches $(\texttt{pp}, \texttt{vp})$ whenever the rule is applied to the copula-PP combination. Notice that when (ID) applies to the term **be** $\big($**on** $x\big)$ **Xavi** while analyzing the LLF in (31a), the variable x is already instantiated at that time. Notice that the rule (ID) comes close to (AUX) as both treat certain lexical terms as identity functions.

ID

$$[\overrightarrow{M}] : \textbf{be}_\alpha\ P : [\overrightarrow{C}] : \mathbb{X}$$

$$[\overrightarrow{M}] : P : [\overrightarrow{C}] : \mathbb{X}$$

$$\alpha \in \{(\texttt{vp}, \texttt{vp}_{adj}), (\texttt{pp}, \texttt{vp})\}$$

Another type of constructions where the copula (with the indefinite determiner) is omitted are similar to (32) (Heim and Kratzer, 1998, Ch. 4). Also interpreting the indefinite determiner as vacuous is somewhat unwanted because this brings the lexical ambiguity of the frequently occurring lexical entry such as the indefinite determiner. Below we show that this ambiguity is unnecessary to extract the simple semantics from the sentences similar to (32).

Xavi is an intellignet football player (32)

a $\big($**intellignet** $\big($**football player**$\big)\big)\big(\lambda x.$**be** x **Xavi**$\big)$ (32a)

The captina of FC Barcelona is Xavi (33)

the $\big($**captain** $\big($**of FC_Barcelona**$\big)\big)\big($**be Xavi**$\big)$ (33a)

All football players are athletes (34)

all $\big($**football player**$\big)\big(\lambda x.$**s athlete** $\big(\lambda y.$**be** $y\ x\big)\big)$ (34a)

We offer two rules, (BE_1) and (BE_2), for simplifying the semantics of the LLFs of the sentences involving the NP-copula-NP triplet. The rules differ only in terms of the instantiation of NP arguments. When the first argument of **be** is instantiated, (BE_1) is applicable and when the second argument is instantiated (BE_2) is applicable. For example, the rule (BE_2) is used while decomposing the LLF in (32a) and (BE_1) for the LLF in (33a). The rules are not limited to indefinite NPs, they can also be used for the definite descriptions (33a) and plurals (34a). The usage of (BE_2) for the indefinite and plural NPs is demonstrated by the tableau in Figure 4.8.

BE_1

$$[\overrightarrow{M}] : q_{\texttt{n},\texttt{vp},\texttt{s}}\ N(\textbf{be}\ c) : [\,] : \mathbb{X}$$

$$[\overrightarrow{M}] : N : c_e : \mathbb{X}$$

$$q \in \{\textbf{a}, \textbf{the}, \textbf{s}\}$$

BE_2

$$[\overrightarrow{M}] : q_{\texttt{n},\texttt{vp},\texttt{s}}\ N(\lambda x.\textbf{be}\ x\ c) : [\,] : \mathbb{X}$$

$$[\overrightarrow{M}] : N : c_e : \mathbb{X}$$

$$q \in \{\textbf{a}, \textbf{the}, \textbf{s}\}$$

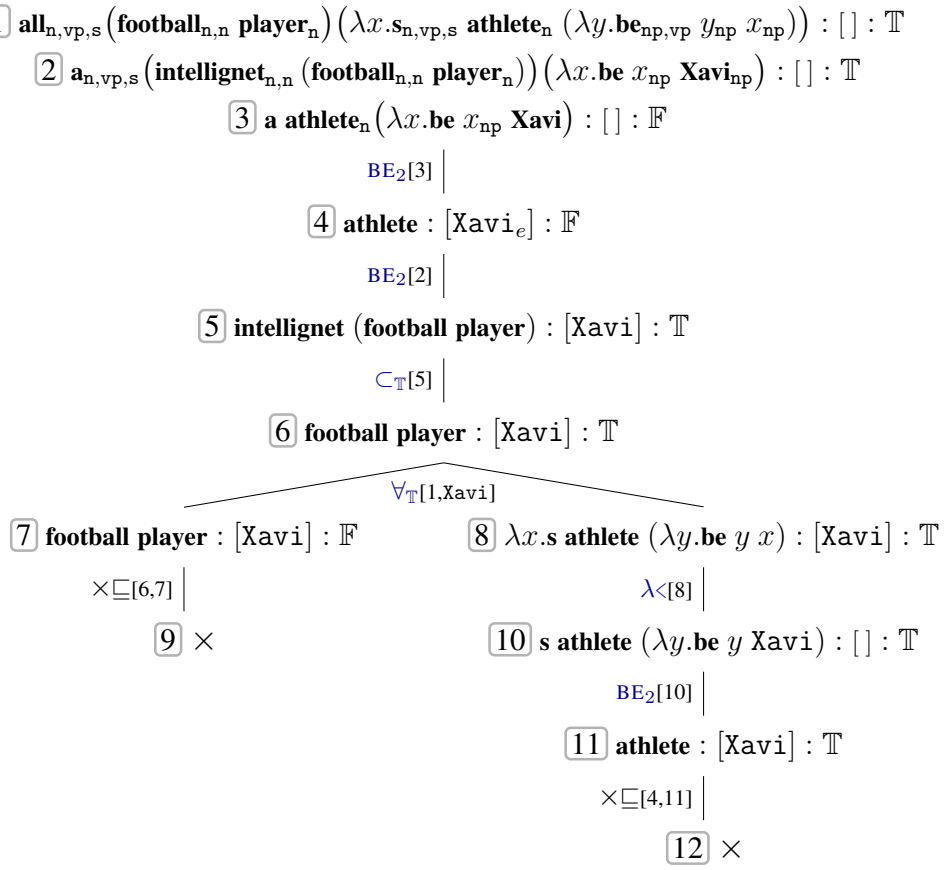
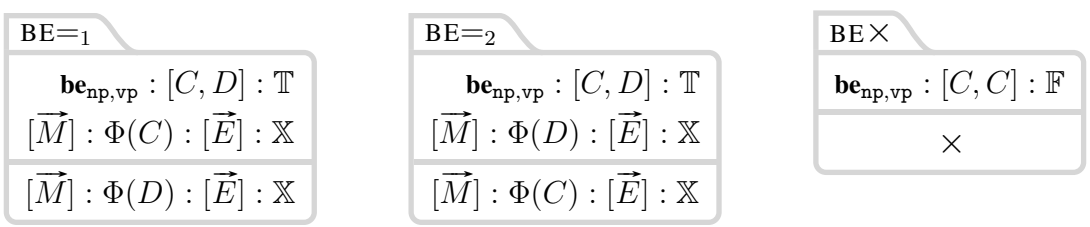


Figure 4.8: The tableau proof demonstrates the usage of the rule (BE$_2$) and shows that "*all football players are athletes*" (34) and "*Xavi is an intellignet football player*" (32) entails "*Xavi is an athlete*".

The rules (BE$_1$) and (BE$_2$) can be seen redundant if we treat **be** as the equality relation with the corresponding (BE$=_1$), (BE$=_2$) and (BE$\times$) rules.[32] Using this treatment, the tableau proof in Figure 4.8 would develop as follows. First, based on 2, an entity p is introduced that is **intellignet (football player)**, hence **football player**, and is equal to Xavi. Then due to 1, an entity a is introduced that is **athlete** and is equal to p. Since a is **athlete**, according to 2 a is not equal to Xavi. The contradiction in the deduced facts, i.e. **be** p Xavi, **be** a p and the negation of **be** a Xavi, is further identified by the equality rules (BE$=_1$), (BE$=_2$) and (BE$\times$).



[32] These rules mirror the first-order logic rules for the equality relation (Fitting, 1990, Ch. 8). In fact, the symmetry and transitivity properties of the equality can be captured by the replacement (modeled by (BE$=_1$) and (BE$=_2$)) and reflexivity (modeled by (BE$\times$)) properties. From a computational point of view, (BE$=_1$) and (BE$=_2$) are inefficient as they often drastically increase the choice points of rule applications during the proof procedure.

Comparing the latter tableau scenario, which introduces two fresh constants, and the one depicted in Figure 4.8 shows that the rules (BE$_1$) and (BE$_2$) offer much more simple and intuitive poofs than treating the copula as equality. This is the main motivation for introducing these rules and hence avoiding the usage of (BE$=_1$) and (BE$=_2$) when it is possible. Unfortunately, it is not always the case that we can avoid (BE$=_1$) and (BE$=_2$). There are entailments, like PROB-9, that need these rules to be proved.

Xavier Hernándes Creus is Xavi Xavi is the captain of FC Barcelona
Xavier Hernándes Creus is a player of FC Barcelona

GOLD: ent, PROB-9

In this section we have given various rules for the copula "*be*". The rule (ID) treats the copula as vacuous with predicative adjectives and PPs. For efficient theorem proving, we introduced (BE$_1$) and (BE$_2$) that help to decompose certain constructions with copula without introducing unnecessary fresh constants or using the inefficient rules (BE$=_1$) and (BE$=_2$) for the equality.

4.6 Rules for passives

The passive constructions occur quite regularly in the open-domain text and due to this reason they are also often included in RTE datasets. Therefore, it is important to account for this syntactic construction in the natural tableau. In this section we give the rules for the passives that are supplemented with the by-phrase.

Since LLFs are automatically generated from the CCG derivation trees, the way the CCG parsers analyze the passive constructions is crucial for the passives rules. Regardless of passives, the by-phrases are PPs and the parsers assign common PP categories to them. In the CCGbank (Hockenmaier and Steedman, 2007), $VP_{pss}\backslash VP_{pss}$ and $(VP_{pss}\backslash VP_{pss})/NP$ are considered as gold categories for the passive by-phrases and the preposition "*by*", respectively.[33] Unsurprisingly, the parsers also do mistakes while analyzing the passive by-phrases, e.g., sometimes they are analyzed as prepositional complements of category PP.[34]

The main idea behind the rules for passives is to identify a passive construction with its corresponding by-phrase and to recover the underlying argument structure. As a by-phrase is itself a PP, the rules for PPs, in particular (M$>$) and (M$<$) from §2.3.2 and (V@PP) from §4.2.2.2, are applicable to it. For convenience we restate these rules below.

[33] The choice is made due to the optional character of the by-phrase (Hockenmaier, 2003, p. 51). The decision also reduces lexical ambiguity and avoids data spareness to some extent (Hockenmaier and Steedman, 2007, Sec. 6)

[34] Moreover, the C&C parser is more detailed with respect of the category features than EasyCCG: the passive by-phrase gets $VP_{pss}\backslash VP_{pss}$ as a gold category from C&C and the featureless $VP\backslash VP$ category from EasyCCG. Unlike the category from C&C, the EasyCCG category does not necessarily link the phrase to the passive constructions. While designing the tableau rules, we also take the ambiguity caused by the EasyCCG category into account.

M<

$$[\overrightarrow{M}] : A\ H : [\overrightarrow{C}] : \mathbb{X} \quad \Rightarrow \quad [\overrightarrow{M}, A] : H : [\overrightarrow{C}] : \mathbb{X}$$

M>

$$[\overrightarrow{M}, A] : H : [\overrightarrow{C}] : \mathbb{X} \quad \Rightarrow \quad [\overrightarrow{M}] : A\ H : [\overrightarrow{C}] : \mathbb{X}$$

V@PP

$$[\overrightarrow{M}] : V_{\mathtt{pp},\alpha}\ (p^{\mathtt{IN}}_{\mathtt{np},\mathtt{pp}}\ D) : [\overrightarrow{C}] : \mathbb{X} \quad \Rightarrow \quad [\overrightarrow{M}] : p^{\mathtt{IN}}_{\mathtt{np},\alpha,\alpha}\ D\ V_{\alpha} : [\overrightarrow{C}] : \mathbb{X}$$

$$\alpha = (\mathtt{np}^{*}, \mathtt{vp})$$

According to our approach in § 4.2.2.2, all PPs that are arguments of a VP are uniformly treated as VP modifiers. Hence a by-phrase, regardless of being a VP modifier or its argument, is also rendered as a VP modifier. If a by-phrase is modifying a verbal term in passive (indicated by the type feature pss), then it is reliable to regard the phrase as a supplement of the passive construction, i.e. as it encoding a subject of the corresponding active construction.[35] The rule (PSS) identifies such a by-phrase with its antecedent and expands the argument structure of the verbal term v with the argument D provided by the phrase. The type β is obtained from α by replacing the final type $\mathtt{s}_{pss}$ with $\mathtt{vp}_{dcl}$, i.e. inserting an additional type np as the last argument in α and changing the passive feature of the final sentence type into declarative.

PSS

$$[\overrightarrow{M}] : \mathbf{by}_{\mathtt{np},\alpha,\alpha} D\ v_{\alpha} : [\overrightarrow{C}] : \mathbb{X} \quad \Rightarrow \quad [\overrightarrow{M}] : v_{\beta} : [\overrightarrow{C}, D] : \mathbb{X}$$

$$\alpha = (\overrightarrow{\gamma}, \mathtt{s}_{pss}) \text{ and } \beta = (\overrightarrow{\gamma}, \mathtt{vp}_{dcl})$$

The rule (PSS) helps not only to reason over the LLFs with passives but, combined with the rules for prepositions, e.g., (VP@PP), it also correctly interprets certain LLFs with erroneous passive analyses. For the demonstration, consider SICK-4386 and the corresponding derivations from EasyCCG. In the premise, the parser wrongly analyzes the by-phrase as a verb complement. However the entailment relation between the sentences is still proved (see Figure 4.12 in Appendix C) with the help of (PSS) and (VP@PP).

A shoe is being $\big[\text{tied}_{VP_{pss}/PP}\ [\text{by}_{PP/NP}\ [\text{a man}]_{NP}]_{PP}\big]_{VP_{pss}}$

A man is tying a shoe

EasyCCG; GOLD: ent; SICK-4386

We described how the passives are analyzed by the CCG parsers and according to it designed the rule (PSS). The rule identifies the by-phrase for a passive construction and rewrites the passive LLF into the synonymous active LLF, where the argument structure of the verbal term has been extended by the *subject* argument. As the by-phrases are also PPs, the rules already designed for PPs (in § 4.2) apply to the by-phrases too. We showed how these rules can be useful while reasoning over the LLFs with wrong analyses of passives.

[35] One can think about the sentence like (35) where the by-phrase is modifying the VP but describes the location of the event.

A man was killed by his car (35)

While such passive construction can be seen as ambiguous, we believe that in its intended/primary semantics the by-phrase carries the *real subject* of the construction.

4.7 Attitude verbs

This section discusses a range of transitive verbs that allows clauses as their arguments. In short, we call such verbs the *attitude verbs*. The attitude verbs comprise so-called *implicative verbs* like ones in "*manage to*", "*force somebody to*" and "*fail to*", or *factive verbs* like "*know that*" and "*regret that*". These verbs are interesting as they give rise to certain presuppositions and entailments.[36] First, we discuss the presuppositions and entailments and their relation to those verb constructions that induce them. Depending on the relations, four major properties of the verbs are presented. To account for these properties in the tableau system, we introduce four tableau rules, each rule modeling a single property. The rules subsequently enable the tableau system to prove certain entailments involving the attitude verbs, for instance, the entailment of (37) from (36).

Suárez managed not to fail to make Neymar score a goal (36)

Neymar scored a goal (37)

4.7.1 Entailment properties of attitude verbs

A main difference between a presupposition and an entailment is that the former is necessary for the well-defined meaning of the source sentence while the latter is necessary for the truth of the source sentence. Due to the importance of presuppositions for the meaning, they are able to *escape* from the scope of negation, questions and if-clauses. For example, asserting either of the sentences in (38) and (39) presupposes the truth of the embedded sentence. The verb constructions, or loosely speaking VPs, with this property are called *factives*. The factive VPs are "*forget that*", "*know that*", "*realize that*", "*was glad to*", etc. A VP that regardless of its truth value presupposes the negation of its sentential argument, e.g., "*pretend that*", is considered *counterfactive*.

Benítez forgot that Cheryshev was suspended (38)

Benítez did not forget that Cheryshev was suspended (39)

On the other hand, entailments are more sensitive to the truth values than presuppositions. The negation can cause an entailment to disappear or to change radically. Consider the sentences in (40) and (41). Assertion of (40) entails that Zidane got a red card while (41) does not entail it. Hence the embedded clause did not survive from the negation. Notice that "*Zidane got a red card*" is not *implicature* of (40) as it cannot be canceled by adding "*but Zidane did not get a red card*" as a further comment to (40).[37]

Materazzi caused Zidane to get a red card (40)

Materazzi did not cause Zidane to get a red card (41)

[36] The properties of these verbs we are interested in are explored in Lauri Karttunen's works (Karttunen, 1971, 2012, 2015) inter alia.

[37] Implicatures are related concept to presuppositions and entailments. In contrast to them, implicatives heavily rely on Grice (1975)'s cooperative principles and are characterized by several properties including *re-enforceability* and *cancelability*. The relations between implicatives and presuppositions are in details discussed by Potts (2015).

two-way implicatives		one-way implicatives			
$++ \mid --$	$+- \mid -+$	$++$	$+-$	$--$	$-+$
manage to bother to succeed in	fail to forget to neglect to	cause NP to force NP to make NP to	refuse to prevent NP from keep NP from	be able to attempt to try to	hesitate to

Table 4.1: Constructions with the entailment properties

Nairn et al. (2006) denotes this property of "*cause NP to*" by $++$, meaning that if the embedding context is positively asserted (denoted by the first sign) then the embedded clause is entailed positively (denoted by the second sign). "*force NP to*" and "*make NP to*" are among other VPs that have the $++$ property (see Table 4.1).

Depending on the polarities of the embedding context and the entailed embedded clause, there are additional three $\pm$ properties $+-$, $--$ and $-+$. The examples of the VPs with these properties are listed in Table 4.1.The VPs having only one of those four entailment properties are called *one-way implicatives*.[38] There are also so-called *two-way implicative* VPs that have two compatible properties from those four. For instance, such verb construct is "*manage to*" which has the $++$ and $--$ properties (i.e. the $++ \mid --$ property). Informally speaking it transmits the negation to the embedded clause. In the same vein, the double properties $++ \mid -+$ and $+- \mid --$ can be associated with the above mentioned factive and counterfactive verbs, respectively. The set of verb constructions that have none of these four implicative properties comprise "*hoped to*", "*wanted to*", "*believes that*", "*offer NP to*", etc.[39]

It is important to mention that the implicative verbs usually come with certain presuppositions. For example, "*manage to*" in (42) and (43) presupposes that the subject at least tried to score a penalty. It is exactly this presupposition that blocks the entailment of (43) from (44). Imagine the situation where Neymar is sleeping. This makes (44) true but not (43) as its presupposition is not supported by the situation.

Neymar managed to score a penalty (42)

Neymar did not manage to score a penalty (43)

Neymar did not score a penalty (44)

4.7.2 Rules for attitude verbs

While talking about presuppositions in the context of the tableau system, one important issue arises that can be explianed in terms of the following questions: In particular, is the truth sign $\mathbb{F}$ a metalinguistic or linguistic negation? Do presuppositions escape from the

[38]The reason why the verb phrases with these properties are referred as implicatives is that they also carry implicatures. For example, when "*be able to*" is uttered positively like in "*Neymar was able to score a penalty*", one is inclined to conclude that Neymar indeed scored unless the sentence is continued with the utterance "*but he did not score*".

[39]A valuable collection of lexical resources, collected by the natural language and reasoning group at Stanford University, that consists of the phrases with adjectives, verbs, and verb-noun collocations annotated with the implicative properties can be found at: http://web.stanford.edu/group/csli_lnr/Lexical_Resources

scope of $\mathbb{F}$ or not? Or are (46) and (47) semantically equivalent entries?

$$\textbf{try}_{\mathtt{vp}_{to},\mathtt{vp}}\ \big(\textbf{to}_{\mathtt{vp}_{b},\mathtt{vp}_{to}}\ (\lambda x.\ \textbf{a penalty score}\ x)\big) : [\textbf{Neymar}] : \mathbb{T} \tag{45}$$

$$\textbf{manage}_{\mathtt{vp}_{to},\mathtt{vp}}\ \big(\textbf{to}_{\mathtt{vp}_{b},\mathtt{vp}_{to}}\ (\lambda x.\ \textbf{a penalty score}\ x)\big) : [\textbf{Neymar}] : \mathbb{F} \tag{46}$$

$$\textbf{not}_{\mathtt{vp},\mathtt{vp}}\ \big(\textbf{manage}_{\mathtt{vp}_{to},\mathtt{vp}}\ \big(\textbf{to}_{\mathtt{vp}_{b},\mathtt{vp}_{to}}\ (\lambda x.\ \textbf{a penalty score}\ x)\big)\big) : [\textbf{Neymar}] : \mathbb{T} \tag{47}$$

At this stage, it is simpler to consider $\mathbb{F}$ as a linguistic negation. Moreover, since there is no proper linguistic counterpart of the metalinguistic negation, its introduction in the natural tableau would deviate the system from the project of natural logic. So, we always allow the top level term **not** to change the truth value of a sign.[40] The presuppositions are projected with the help of the tableau rules and they are asserted on a branch like entailments that escape from the $\mathbb{F}$ sign. For instance, if we wish to further analyze the presupposition triggered by "*manage to*", we would design the rule that replaces **manage** with **try** and sets the sign to $\mathbb{T}$, in other words, introduces (45) from (46). By this decision, the tableau system to some extent adheres to *Strawson's entailment* (see §4.3.1.1) in case of the VPs with presuppositions.

It is sufficient to model the verbs with the $\pm$ properties in Table 4.1 by designing the tableau rules that capture these four properties. Then the terms with the number of the properties (e.g., two-way implicatives) will be processed with the same number of the corresponding rules. The verb constructions with the entailment properties are syntactically versatile. To capture these constructions in a compact way, we borrow the meta-symbol "?" from regular expressions that marks terms and argument types as optional. The schema in (48), represents the abstract LLF that matches the attitude verbs. The concrete matched instances of LLFs are presented in (48a–g) along with their corresponding linguistic expressions.

$$h_{?\mathtt{np},\beta,\mathtt{vp}}\ ?C_{\mathtt{np}}\ (?p_{\alpha,\beta}\ V_{\alpha})\big)\ \ \text{s.t.}(\alpha,\beta) \in \{(\mathtt{vp},\mathtt{vp}),(\mathtt{vp},\mathtt{pp}),(\mathtt{s},\mathtt{s})\} \tag{48}$$

$$\textbf{manage}_{\mathtt{vp}_{to},\mathtt{vp}}(\textbf{to}_{\mathtt{vp}_{b},\mathtt{vp}_{to}}\ \textbf{score}_{\mathtt{vp}}) \quad \textit{manage to equalize} \tag{48a}$$

$$\textbf{succeed}_{\mathtt{vp}_{to},\mathtt{vp}}(\textbf{in}_{\mathtt{vp}_{ng},\mathtt{pp}}\ \textbf{score}_{\mathtt{vp}}) \quad \textit{succeed in scoring} \tag{48b}$$

$$\textbf{prevent}_{\mathtt{np},\mathtt{vp}_{to},\mathtt{vp}}\textbf{Pique}_{\mathtt{np}}(\textbf{from}_{\mathtt{vp}_{ng},\mathtt{pp}}\ \textbf{score}_{\mathtt{vp}}) \quad \textit{prevent Pique from scoring} \tag{48c}$$

$$\textbf{glad}_{\mathtt{vp}_{to},\mathtt{vp}_{adj}}(\textbf{to}_{\mathtt{vp}_{b},\mathtt{vp}_{to}}\ \textbf{run}_{\mathtt{vp}}) \quad \textit{glad to win} \tag{48d}$$

$$\textbf{witness}_{\mathtt{np},\mathtt{vp},\mathtt{vp}}\textbf{FC\_Barcelona}_{\mathtt{np}}\ \textbf{win}_{\mathtt{vp}} \quad \textit{witness FC Barcelona win} \tag{48e}$$

$$\textbf{realize}_{\mathtt{s}_{em},\mathtt{vp}}(\textbf{that}_{\mathtt{s},\mathtt{s}_{em}}(\textbf{score}_{\mathtt{vp}}\ \textbf{Xavi}_{\mathtt{np}})) \quad \textit{realize that Xavi scored} \tag{48f}$$

$$\textbf{lie}_{\mathtt{np},\mathtt{s}_{em},\mathtt{vp}}\textbf{Luis}_{\mathtt{np}}(\textbf{that}_{\mathtt{s},\mathtt{s}_{em}}(\textbf{score}_{\mathtt{vp}}\ \textbf{Xavi}_{\mathtt{np}})) \quad \textit{lied Luis that Xavi scored} \tag{48g}$$

After identifying such kind of constructions, we need to check the functional term $h_{?\mathtt{np},\beta,\mathtt{vp}}\ ?C_{\mathtt{np}}$ of (48) on the entailment properties. We assume here that the knowledge about these properties are provided by the signature. For example, in the signature, the $+-$ property of "*prevent Pique*" can be obtained from the type $(\mathtt{np},\{+-\}\mathtt{pp},\mathtt{vp})$ of the **prevent** term recursively, where I, J, K are subsets of the entailment properties $\{++,+-,--,-+\}$:

$$A \text{ has the } i \text{ property} \quad \text{if } A \text{ is of type } (I\alpha, J\beta) \text{ and } i \in I$$

[40] More precisely, the adopted interaction between the lexical negation **not** and the truth sign, does not commit us to interpret both **not** and $\mathbb{F}$ as linguistic negations but as the negations of the same type—either linguistic or metalinguistic. The way the entries with the $\mathbb{F}$ sign are unfolded further determines the type of the negation.

$$A_{I\alpha,J\beta}\ B_{K\gamma} \text{ is of type } I\beta \quad \text{if } \gamma \sqsubseteq \alpha$$

The rules modeling the entailment properties are given below. Besides the requirement for the corresponding entailment property, they also satisfy the shared constraints.[41] The tableau proof for the entailment of (37) from (36) demonstrates the new rules in action in Figure 4.9.

++

$h_{?\mathtt{np},\beta,\mathtt{vp}}\ ?C_{\mathtt{np}}\ (?p_{\alpha,\beta}\ V_\alpha) : [\vec{D}] : \mathbb{T}$

$V_\alpha : [\vec{E}] : \mathbb{T}$

$h\ ?C$ has the property $++$

+−

$h_{?\mathtt{np},\beta,\mathtt{vp}}\ ?C_{\mathtt{np}}\ (?p_{\alpha,\beta}\ V_\alpha) : [\vec{D}] : \mathbb{T}$

$V_\alpha : [\vec{E}] : \mathbb{F}$

$h\ ?C$ has the property $+-$

−+

$h_{?\mathtt{np},\beta,\mathtt{vp}}\ ?C_{\mathtt{np}}\ (?p_{\alpha,\beta}\ V_\alpha) : [\vec{D}] : \mathbb{F}$

$V_\alpha : [\vec{E}] : \mathbb{T}$

$h\ ?C$ has the property $-+$

−−

$h_{?\mathtt{np},\beta,\mathtt{vp}}\ ?C_{\mathtt{np}}\ (?p_{\alpha,\beta}\ V_\alpha) : [\vec{D}] : \mathbb{F}$

$V_\alpha : [\vec{E}] : \mathbb{F}$

$h\ ?C$ has the property $--$

Shared constraints: $(\alpha, \beta) \in \{(\mathtt{vp}, \mathtt{vp}), (\mathtt{vp}, \mathtt{pp}), (\mathtt{s}, \mathtt{s})\}$; $\vec{E}$ is empty iff $\alpha = \beta = \mathtt{s}$; when $\vec{E}$ is nonempty, $\vec{E} = C$ if C appears, otherwise $\vec{E} = \vec{D}$

Notice that the entailment relations licensed by the $\pm$ properties of the attitude verb constructions are decreasing in terms of the length of a sentence: the conclusion is always shorter that the premise. But one might want to capture the entailments, like (50) from (49), without decreasing the length of a conclusion. Certain entailments, like the latter example, can be captured via monotonicity calculus of the tableau system if the monotonicity features are associated with the attitude verbs, e.g., **manage** being $\uparrow$mon in the first argument position. This solution is far from being perfect as it also licenses the unsound entailments like (51) from (49). In general, it requires further analysis of the semantics and pragmatics of the attitude verbs to capture such non-decreasing entailments adequately.

Neymar managed to score a free kick (49)

Neymar managed to score a goal (50)

Neymar managed to shoot a free kick (51)

In order to model additional properties of certain propositional attitude (e.g., "*know*" or "*believe*") and modal verbs in the tableau system, it is sufficient to move from extensional types to intensional ones (e.g., the sentential LLFs will be of type st), to introduce the corresponding reachability relation R over possible worlds in the tableau entries and

[41] Notice that the rules do not restrict the optional lexical term $p_{\alpha,\beta}$. This decision is made in order to keep the rules general, simpler and fewer. In the application, we do not expect p matching a wrong term, but if it happens, it would be an interesting example and the evidence for further constraining the rules.

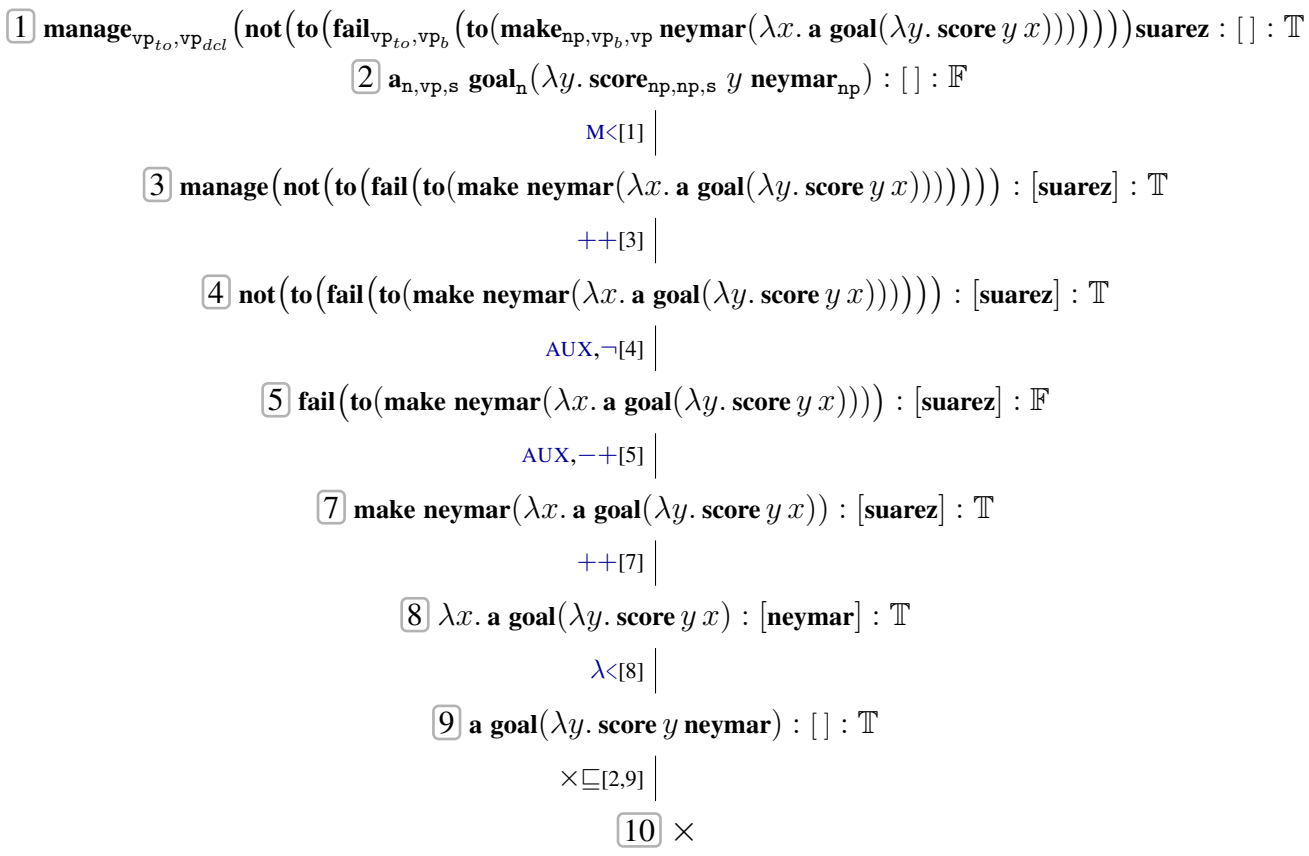


Figure 4.9: The tableau proof demonstrates the rules for the attitude verbs. The following typing is assumed for the rest of the terms: **not**$_{\mathrm{vp},\mathrm{vp}}$, **to**$_{\mathrm{vp}_b,\mathrm{vp}_{to}}$, **suarez**$_{\mathrm{np}}$ and all variables are of type np. The information about the entailment properties of the attitude verbs are found in the signature: **manage**$_{\{++,--\}\mathrm{vp},\mathrm{vp}}$, **fail**$_{\{+-,-+\}\mathrm{vp},\mathrm{vp}}$ and **make**$_{\mathrm{np},\{++\}\mathrm{vp},\mathrm{vp}}$.

to borrow the modal rules from the corresponding propositional modal logics.[42] At this moment, we do not develop a modal reasoning in the natural tableau system as recently there is no RTE dataset which would evaluate a system on that. The section of SICK (Cooper et al., 1996) on attitude verbs (with 13 RTE problems) hardly contains the problems requiring some sort of modal reasoning.

We have presented the tableau rules for the entailment properties associated with the attitude verbs, including the implicative and (counter) factive verbs. The rules serve as a swift and simple device for reasoning over the constructions involving the attitude verbs. At this stage, the coverage of this simple solution is enough for the existing RTE problems. Hence, there is no need to complicate the tableau system and the underlying semantics to account for better reasoning over, e.g., the propositional attitude verbs.

[42] It is not always necessary to state the modal relation R explicitly in tableau entries, there are also so-called *implicit* tableau systems where R and its properties are in some way built into the rules (Goré, 1999).

4.8 Conclusion

We have presented a collection of the rules which account for semantics of various lexical elements and constructions. The rules were collected in a data-driven fashion, using FraCaS and the trial portion of SICK, and cover the following expressions and phenomena: adjectives, auxiliaries, prepositional phrases, particles, definite NPs, expletive "*there*", verb subcategorization, open compound nouns, light verbs, copula, passives and attitude verbs. Since our initial goal is to model those phenomena that occur in textual entailment datasets, at this stage, we do not account for tense and aspect or mood as they are officially ignored in RTE datasets (Dagan et al., 2006). As a result, we do not treat temporal expressions in a special way but as ordinary PPs.

Some of the presented tableau rules do more than unfolding semantics of certain LLFs. The rules for prepositional phrases are such. Apart from extracting semantics, they also tackle the problem of PP attachment. Since the parsers often make mistakes with respect to the type and the site of PP attachment, we adopted special rules that treat PP attachments as ambiguous and consider several scenarios for them. Theoretically, such rules lead to unsound inferences, but as it turns out later (Chapter 6), the considered RTE datasets cannot reveal this unsoundness. If the parsers were flawless in PP attachments, then we could simply ignore these rules. So, the introduced unsoundness is not a drawback of the tableau system per se.[43]

For the rest of the rules we would like to emphasize the following. The definite descriptions are treated as referring expressions as this is their main function in the RTE problems. The rules for open compound nouns and light verb constructions are economical in the sense that they do not make up new compounds or constructions. They identify only contradictions with respect to these phenomena, e.g., c_e being "*bottle for beer*" but not "*beer bottle*". The rules for verb subcategorization are redundant when using event semantics but they are efficient from a theorem proving perspective. Due to this reason, the rules will be included in an implemented tableau theorem prover (Chapter 5). In the next chapter, we present an implemented theorem prover which is faithful to the natural tableau system. The rule inventory of the prover will mostly constitute the rules presented in this chapter.

[43] While talking about the soundness, on might ask a question concerning the completeness of the tableau system. Of course, the tableau system presented in this thesis is not complete, and that is not surprising. Our aim is not to have a complete proof system for some fragment of natural logic but to have a sound and incomplete proof system that covers as much linguistic constructions as possible.

Appendix C

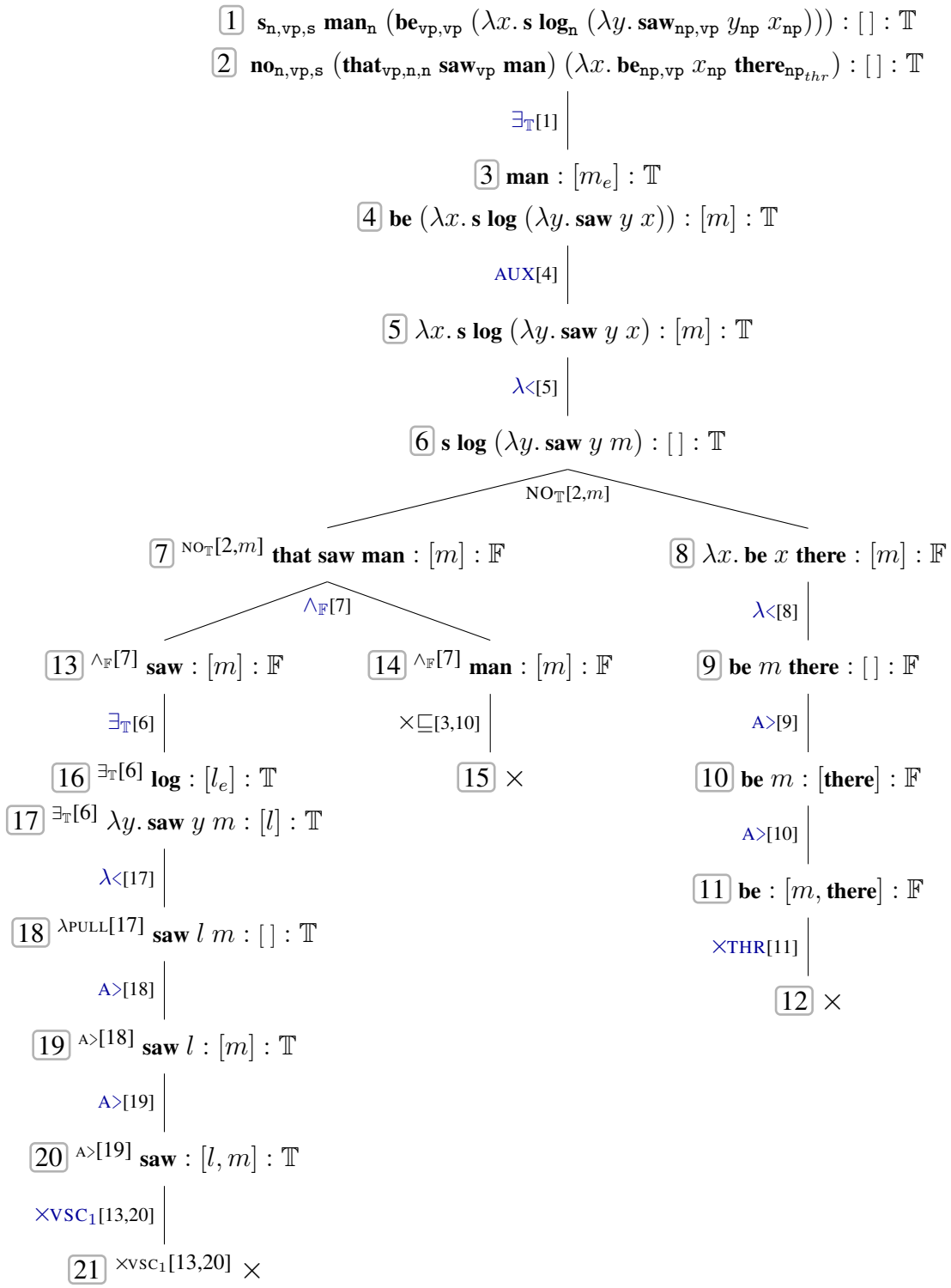


Figure 4.10: The closed tableau proves 1417 as contradiction.

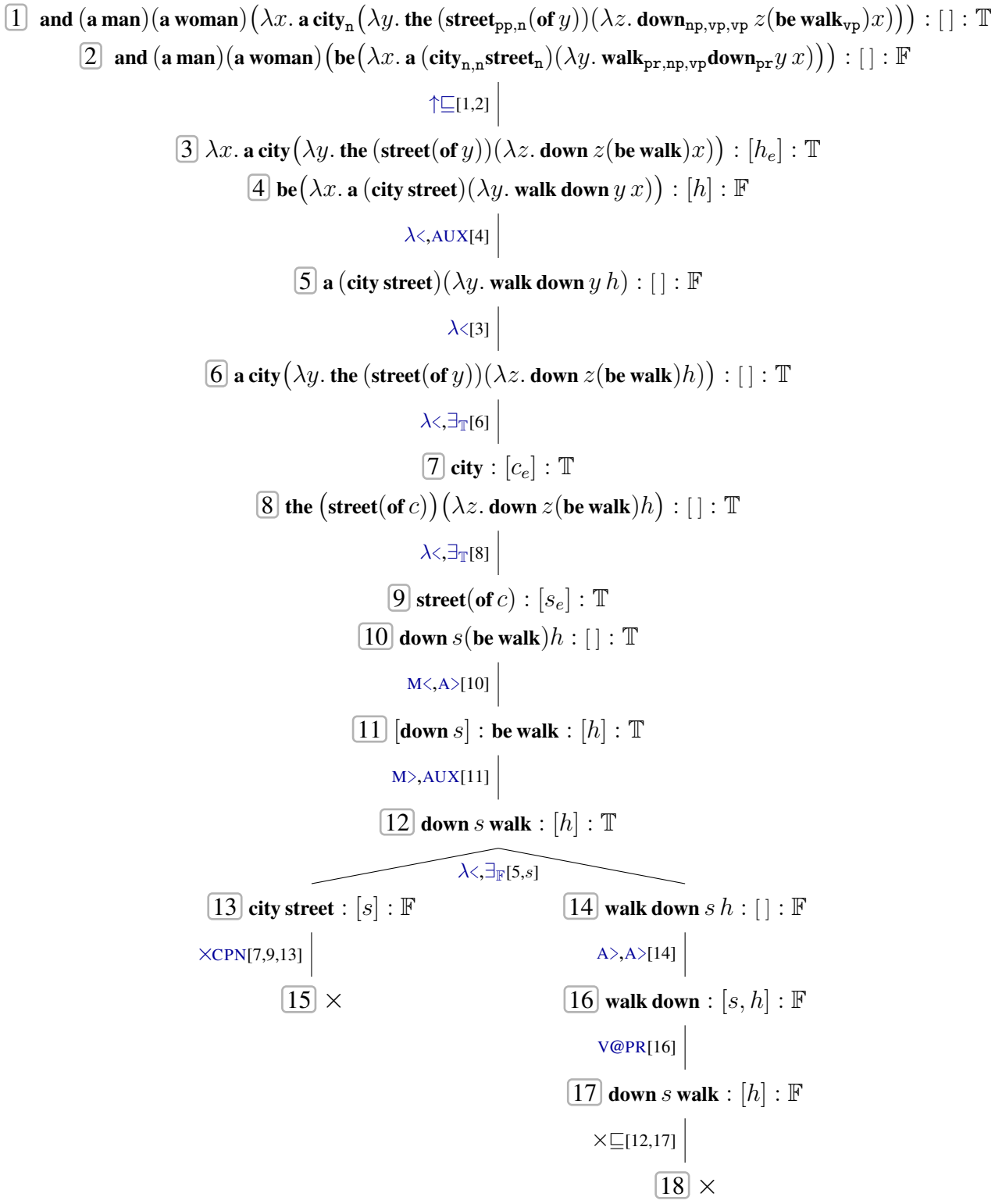


Figure 4.11: The closed tableau classifies SICK-7755 as contradiction. Due to lack of space, the following information about typing was omitted in the tableau: $\mathbf{man}_{\mathtt{n}}$, $\mathbf{woman}_{\mathtt{n}}$, $\mathbf{and}_{(\mathtt{vp},\mathtt{s}),(\mathtt{vp},\mathtt{s}),\mathtt{vp},\mathtt{s}}$, $\mathbf{a}_{\mathtt{q}}$, $\mathbf{the}_{\mathtt{q}}$, $\mathbf{of}_{\mathtt{np},\mathtt{vp},\mathtt{vp}}$, $\mathbf{be}_{\mathtt{vp},\mathtt{vp}}$ and all variables are of type np. The rule applications are written in short; for example, ($\lambda<$,$\exists_{\mathbb{T}}$[6]) means that first ($\exists_{\mathbb{T}}$) is applied to the node 6 and then ($\lambda<$) is applied to one of the resulted consequent nodes. The left branch that appears after the ($\uparrow\sqsubseteq$[1,2]) rule application is omitted as it closes instantly.

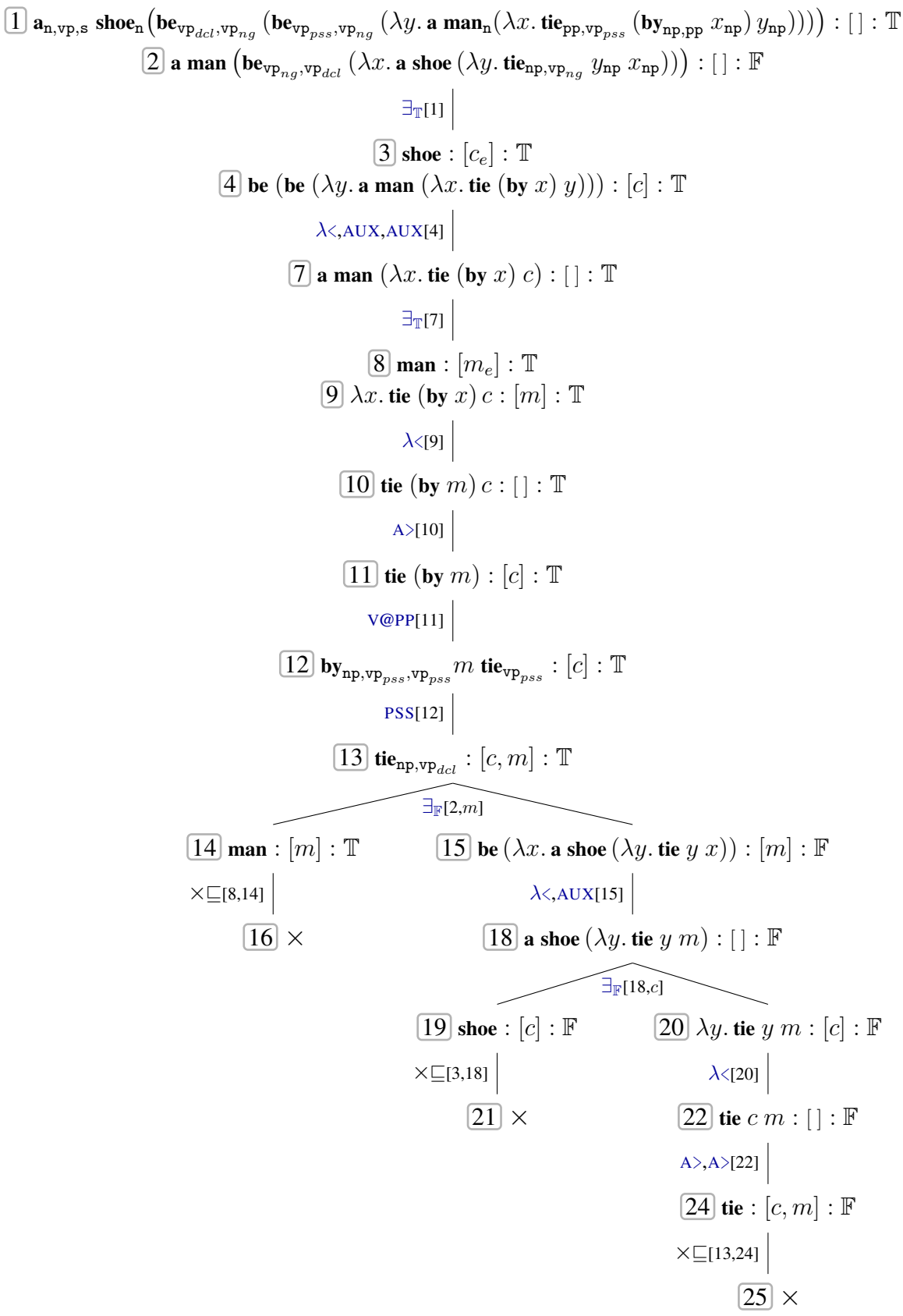


Figure 4.12: Despite the wrong derivation for the premise returned by EasyCCG, the tableau system proves SICK-4386 as entailment with the help of (PSS) and (VP@PP).

Chapter 5

Theorem Prover for Natural Language

In this chapter we describe a theorem prover that is based on the natural tableau system. It acts like a theorem prover for natural language, i.e. takes natural language expressions as an input and can provide judgments about the semantic relations between them. The prover consists of three main components: a syntactic parser, the generator for Lambda Logical Forms (LLFs) and a sub-prover for the adopted version of natural logic (see Figure 5.1). The parser component and the generator were already discussed in Chapter 3. The sub-prover for natural logic, called NLogPro, itself involves four components: a signature, a proof engine, an inventory of tableau rules and a knowledge base. We go through each component of the sub-prover and describe them in details. First, the extraction of lexical semantic relations from WordNet is presented. Since tableau theorem proving is all about rule applications, we study four properties of tableau rules that are relevant from a computational point of view. Based on these properties we define efficiency criteria where each criterion induces an efficiency order over the rules. As it turns out later, certain efficiency orders lead to shorter tableau proofs. In addition to the efficiency orders, we also introduce additional tableau rules to encourage short proofs. The proof engine (PE) is a component which is directly responsible for building tableau proofs. Given a tableau branch, the engine applies the most efficient rule among all applicable ones, where the efficiency is defined by an efficiency criterion.

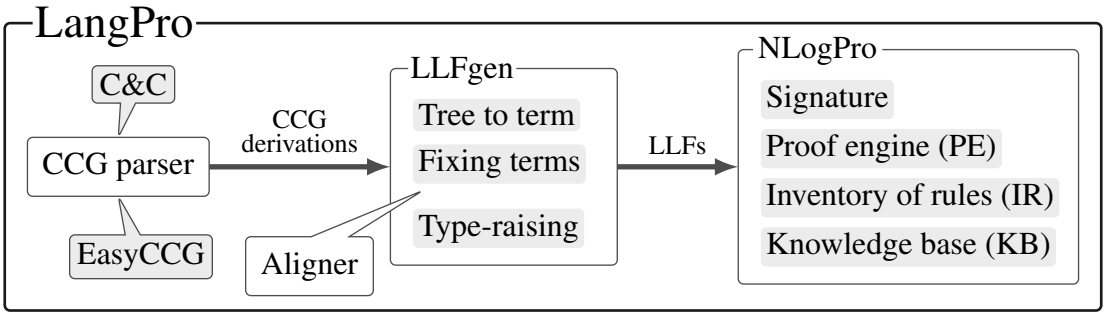


Figure 5.1: The architecture of a natural language theorem prover, called LangPro.

After describing each individual component of NLogPro, we return to theorem provers. In particular, first we introduce the architecture and the functionality of NLogPro. We also take a closer look at the PE and describe how it builds actual proofs. In the end, we present the final theorem prover which operates on natural language expressions. We describe its

functionality on a sample entailment problem. For the final prover and its sub-prover, we give the pseudocodes describing their underlying algorithms. Since the tableau rules are specially designed for natural language expressions and the implemented prover reasons in natural language, we regard it as the natural language prover, shortly LangPro.[1]

5.1 Knowledge base

Textual entailments usually presuppose some kind of knowledge and capturing the latter is one of the main challenges of Recognizing Textual Entailment (RTE). Lexical knowledge represents one of such wanted knowledge. Fortunately, there exists WordNet (Fellbaum, 1998)—a high quality lexical database for natural languages. The section describes how semantic relations over lexical terms can be extracted from WordNet. First, we define semantic relations over word senses via certain WordNet relations. Then we give two main approaches for transferring the semantic relations over word senses to the relations over lexical terms. The transfer is not trivial as in general a single lexical term might have several word senses. The approaches represent attempts to overcome the problem of word sense disambiguation in textual entailments. Notice that later we will use the Knowledge Base (KB) solely based on WordNet. This will be sufficient to obtain competitive results on certain RTE datasets (§6.4).

WordNet is a lexical database which covers the open-class words, i.e. nouns, verbs, adjectives and adverbs (also including the phrases that correspond to atomic concepts).[2] It contains information about word senses and various conceptual and lexical relations between them. Word senses are grouped into sets of cognitive synonyms, called *synsets*, which are annotated with glosses and sometimes with short sample sentences. For instance, a synset which consists of only two senses cry^2_v and $weep^1_v$ has the gloss "*shed tears because of sadness, rage, or pain*".[3] At the same time, "*cry*" has other 11 senses (five nouns and six verbs) while "*weep*" has only a single sense.

WordNet provides around 20 binary relations, where small number of them are *conceptual relations*, defined over synsets, and the rest are *lexical relations* defined over word senses. Below we describe several conceptual and lexical relations. The $\text{Hyponymy}_{\text{WN}}$ conceptual relation links more specific synsets to more general ones, e.g., a synset $\{sob^1_v\}$ is a hyponym of $\{cry^2_v, weep^1_v\}$. The relation is defined for noun and verb synsets. The reversed version of $\text{Hyponymy}_{\text{WN}}$ is called $\text{Hypernymy}_{\text{WN}}$. The $\text{Similarity}_{\text{WN}}$ conceptual relation holds only for the adjective synsets that have similar meanings: $\{huge^1_a, immense^1_a, vast^1_a\}$ is similar to $\{big^1_a, large^1_a\}$ and vice versa. The $\text{Instance}_{\text{WN}}$ conceptual relation for the noun synsets holds when one synset represents an instance of another, e.g., $\{Tbilisi^1_n, Tiflis^1_n, capital\,of\,Georgia^2_n\}$ is an instance of $\{national\,capital^1_n\}$). $\text{Antonymy}_{\text{WN}}$ is a lexical relation and holds for the opposite word senses, e.g., $\langle weep^1_v, laugh^1_v\rangle$ and $\langle back^2_r,$

[1] An online demo of LangPro is available at http://www.naturallogic.pro/langpro/

[2] WordNet is accessible online at http://wordnetweb.princeton.edu/perl/webwn

[3] We employ the adopted convention in NLP community while presenting word senses. In particular, each word sense is denoted by w^i_p where w stands for a word, p for the part of speech (i.e. n for nouns, v for verbs, a for adjectives and r for adverbs) and i for the sense number of w_p. It is important to note that the senses for w_p are numbered according to their frequency in SemCor (Miller et al., 1993). For example, cry^2_v is the second sense of the verbal "*cry*" while its first sense cry^1_v, the most frequent one, means to "*utter a sudden loud cry*".

$\textit{forward}_r^1\rangle$ are elements of the antonymy relation. Another lexical relation is $\text{Derivation}_{\text{WN}}$. It connects the word senses that have word forms related with derivational morphology, e.g., $\textit{protect}_v^1$ is in the derivation relation with $\textit{protection}_n^1$, $\textit{protection}_n^2$, $\textit{protector}_n^1$ and $\textit{protective}_a^1$. Both $\text{Antonymy}_{\text{WN}}$ and $\text{Derivation}_{\text{WN}}$ are symmetric relations by definition.

On the way to define the semantic relations for lexical terms, we first describe the semantic relations for word senses. In other words, given the WordNet relations over synsets or senses, we suggest the semantic relations like inclusion ($\sqsubseteq$), exclusion ($|$), exhaustion ($\smile$) and instance ($\prec$) for word senses. It is obvious that $\text{Hyponymy}_{\text{WN}}^{\text{TR}}$, the reflexive and transitive closure of $\text{Hyponymy}_{\text{WN}}$, is a good candidate for defining semantic inclusion ($\sqsubseteq_S$) over senses. Since $\text{Hyponymy}_{\text{WN}}$ is defined only for noun and verb synsets, we adopt $\text{Similarity}_{\text{WN}}$ for adjective synsets. In particular, for two senses S_1 and S_2 we define $S_1 \sqsubseteq_S S_2$ iff there are synsets SS_1 and SS_2 such that $S_1 \in SS_1$, $S_2 \in SS_2$ and SS_1 is either in $\text{Hyponymy}_{\text{WN}}^{\text{TR}}$ or $\text{Symilarity}_{\text{WN}}$ relation with SS_2. The $\text{Instance}_{\text{WN}}$ relation is used to define the instance relation ($\prec_S$) for noun senses in the similar way: $S_1 \prec_S S_2$ iff $S_1 \in SS_1$, $S_2 \in SS_2$ and SS_1 is in $\text{Instance}_{\text{WN}}$ relation with SS_2.

The rest of the semantic relations, exhaustion ($\smile$) and exclusion ($|$), do not have analogous relations in WordNet. While it is not clear how the exhaustion relation can be extracted from WordNet, there are several ways to obtain the exclusion relation from it. The word senses in the $\text{Antonymy}_{\text{WN}}$ relation are excellent candidates for disjoint senses: $S_1|_S S_2$ iff $S_1 \in SS_1$, $S_2 \in SS_2$ and SS_1 is in $\text{Antonymy}_{\text{WN}}$ relation with SS_2. The exclusion relation over senses can be further augmented with co-hyponymy relation, i.e. let $S_1|_S S_2$ if $S_1 \in SS_1$, $S_2 \in SS_2$ and SS_1 and SS_2 have the same direct hypernym. In this way, $\textit{bookcase}_n^1|_S\textit{table}_n^2$ as their synsets are direct hyponyms of the synset consisting of $\textit{furniture}_n^1$, $\textit{piece of furniture}_n^1$ and $\textit{article of furniture}_n^1$. Alternatively, the exclusion relation can be made even more general by adding the pairs of noun senses for which no semantic relation holds (e.g., this general approach is adopted by MacCartney (2009)). Notice that both augmentations of exclusion for senses introduce unsound pairs. For example, in the case of co-hyponyms, $\textit{puppy}_n^1$ and $\textit{poodle}_n^1$ are falsely considered disjoint as their corresponding synsets are co-hyponyms dominated by the synset of $\textit{dog}_n^1$. Such unsound lexical knowledge can be the source of wrong proofs.

In the previous paragraph, we have discussed how the WordNet relations can be used to define the semantic relations over word senses. Now we need to transfer these relations from word senses to lexical terms. One straightforward solution is to use a Word Sense Disambiguation (WSD) system for identifying the correct sense of a word in a given context. Unfortunately, WSD is itself an open problem in Natural Language Processing (NLP) as the top WSD systems slightly outperform simple baseline approaches.[4] Plausibly this is the main reason why RTE systems usually do not use WSD systems for inference (Dagan et al., 2013, p. 125).

In the current work, we do not employ any existing WSD system for two reasons. First, we would like to keep the architecture of the theorem prover simple, and second, it is interesting to find out the upper limits of our method on RTE without any external WSD

[4]The upper limit for WSD systems with respect to the WordNet senses, so-called *fine-grained senses*, is not high. Based on three challenges in WSD, the inter-annotator agreement for *all-words*, i.e. assigning a WordNet sense to each open-class word of a sentence, is between 67% and 80% (Navigli, 2009, p. 43). Moreover, WSD systems do not show big improvement over the baseline, which assigns most frequent senses to words. Usually difference in accuracy between them is less than 10% and varies based on the domain of text. For more details on WSD, we refer to Navigli (2009).

TEP	Premise/Text	Conclusion/Hypothesis	MS	MFS
PROB-10	The child is crying	The child is weeping	N	E
PROB-11	The child is crying	The child is screaming	E	E
PROB-12	The child is crying	The child is screaming and weeping	N	E
PROB-13	The child is screaming	The child is weeping	N	N

Table 5.1: Comparison of the MS and MFS approaches in the natural tableau. N and E stand for *neutral* and *equivalent* (i.e. bi-directional entailment) respectively.

system. We propose two main approaches how to link the relations over word senses to those over lexical terms. The first approach, so-called the *multi-sense* approach, assumes that each lexical term has multiple senses independently from the context it occurs in. According to it, a semantic relation R holds between the lexical terms $A_{\mathtt{a}}$ and $B_{\mathtt{b}}$ (i.e. $A_{\mathtt{a}}\,\mathcal{R}\,B_{\mathtt{b}}$ is in the KB) iff each of them has at least one WordNet sense, $A^i_{\mathtt{a}}$ and $B^j_{\mathtt{b}}$ respectively, such that $A^i_{\mathtt{a}}\,\mathcal{R}_S\,B^j_{\mathtt{b}}$ holds, where $\mathcal{R}_S$ is a relation over senses defined in terms of the WordNet relations and `a` and `b` indicate a part of speech. For instance, $\mathbf{cry}_{\mathtt{vp}}$ and $\mathbf{weep}_{\mathtt{vp}}$ are equivalent, i.e. $\mathbf{cry}_{\mathtt{vp}} \sqsubseteq \mathbf{weep}_{\mathtt{vp}}$ and $\mathbf{weep}_{\mathtt{vp}} \sqsubseteq \mathbf{cry}_{\mathtt{vp}}$, as they have the synonymous senses cry^2_v and $weep^2_v$. Similarly, the lexical terms A and B are disjoint iff they have antonymous senses: $\mathbf{empty}_{\mathtt{n,n}}|\mathbf{full}_{\mathtt{n,n}}$ holds because $empty^1_a|_S full^1_a$. While using the multi-sense approach, one might restrict a set of senses per word or allow all the senses of a word. The latter we call the *all-senses* approach.

The second approach employs a single sense for each word. In the case of assigning the most frequent sense to each word, we get the *first-sense* approach: $A_{\mathtt{a}}\,\mathcal{R}\,B_{\mathtt{b}}$ is in the KB iff $A^1_{\mathtt{a}}\,\mathcal{R}_S\,B^1_{\mathtt{b}}$ holds. A WSD task with all-words and WordNet senses (Navigli, 2009) show that it is challenging for WSD systems to outperform a baseline model that assigns most frequent senses. Therefore, we think it is worthy to test the first-sense approach on an RTE task. If we assume that each word in a context can be unambiguously tagged with a sense, then the gold/correct sense assignment is a member of the space of single-sense approaches.

It is clear that the tableau system with all-senses is stronger than with the any single-sense approach (see Table 5.1): if some relation holds for single-sense it automatically holds for all-senses. For example, the natural tableau system using all-senses proves PROB-10 and PROB-11 in Table 5.1 as equivalent because it considers all the senses of "*cry*" in each proof. With all-senses it is also possible to prove PROB-12 as equivalent.[5] On the other hand, the first-sense approach only proves PROB-11 as equivalent because the most frequent senses of "*cry*" and "*scream*" are synonymous and are not related to $weep^1_v$. Notice that based on all-senses, PROB-13 is not even entailment while PROB-10 and PROB-11 encode equivalences—no senses of "*weep*" and "*scream*" are in the hyponymy or synonymy relations. This *illogical* behavior is caused by the sense indeterminism (i.e. underspecification) of the all-senses approach. All in all, while the first-sense approach looks kind of poor (e.g., it fails to prove PROB-10), the all-senses approach is too powerful than it is necessary for natural reasoning. For these reasons, we think both methods are worth of trying with the natural tableau.

[5]Notice that no single-sense approach is able to prove entailment in PROB-12. Hence, the all-senses approach is even stronger than the space of single-sense approaches taken together.

We have discussed the population of the KB with the WordNet relations. The inclusion and instance relations are naturally extracted from WordNet, but there is no direct way of obtaining the exhaustion and exclusion relations from it. While jointly exhaustive pairs of words are not common, there are many word pairs in the exclusion relation and WordNet's antonymy relation seems too little for modeling it. Future improvements of the exclusion relation in the KB should await further advances in modeling (in)compatibility of words, which was recently tackled by Kruszewski and Baroni (2015). We have presented two general methods, multi-sense and single-sense, for transforming the relations over word senses into the relations over lexical terms. The approaches do not use any WSD methods. This will allow us to estimate exclusively the natural tableau system and WordNet for wide-coverage reasoning. Moreover, taking into account the problem of knowledge sparsity for rule-based RTE systems, we believe the multi-sense approach might seem an interesting option for the tableau prover to overcome the problem to some extent.

5.2 Inventory of the rules

The inventory of tableau rules (IR) is the most important component for the natural tableau system. It contains the inference rules, representing the simplest inference steps, which underlie reasoning over LLFs. The IR consists of the rules discussed in the previous chapters, excluding the deprecated rules with a dark gray background. In this subsection, we discuss the properties of tableau rules that are crucial for theorem proving and, as we will see later in § 5.3, are used to guide the rule application process in the implemented tableau theorem prover. To have a smart strategy for rule applications and get shorter tableau proofs, we add *derivable rules* to the IR. From the theoretical point of view, the derivable rules do not contribute to the natural tableau proof system per se but they can guide the theorem prover to shorter proofs. This contribution of the derivable rules is crucial from the application point of view.

5.2.1 Properties of the rules

The tableau rules are diverse according to their function and structure. Some of them are branching rules or introduce a fresh entity in a tableau branch. We identify several properties and based on them define an efficiency feature vector for each tableau rule. The vectors are then used to compare tableau rules for computational efficiency. Below we list the properties of tableau rules that are crucial for finding short tableau proofs:

Branching A tableau rule is either *branching* or *non-branching* depending on the number of its consequent branches. For instance, ($\exists_{\mathbb{F}}$) is a branching rule while ($\exists_{\mathbb{T}}$) is non-branching (some of the rules are presented in Figure 5.2 for convenience). In general a branching rule is more efficient than a non-branching one because usually two branches yield more options of rule application. But sometimes applying a branching rule prior to a non-branching one might lead to a smaller closed tableau or it can be a strategy for quickly finding an open branch.

Semantic equivalence Whether the antecedents of a tableau rule is semantically equivalent to or stronger than the consequents, the rule is *semantic equivalence* or *non-equivalence*

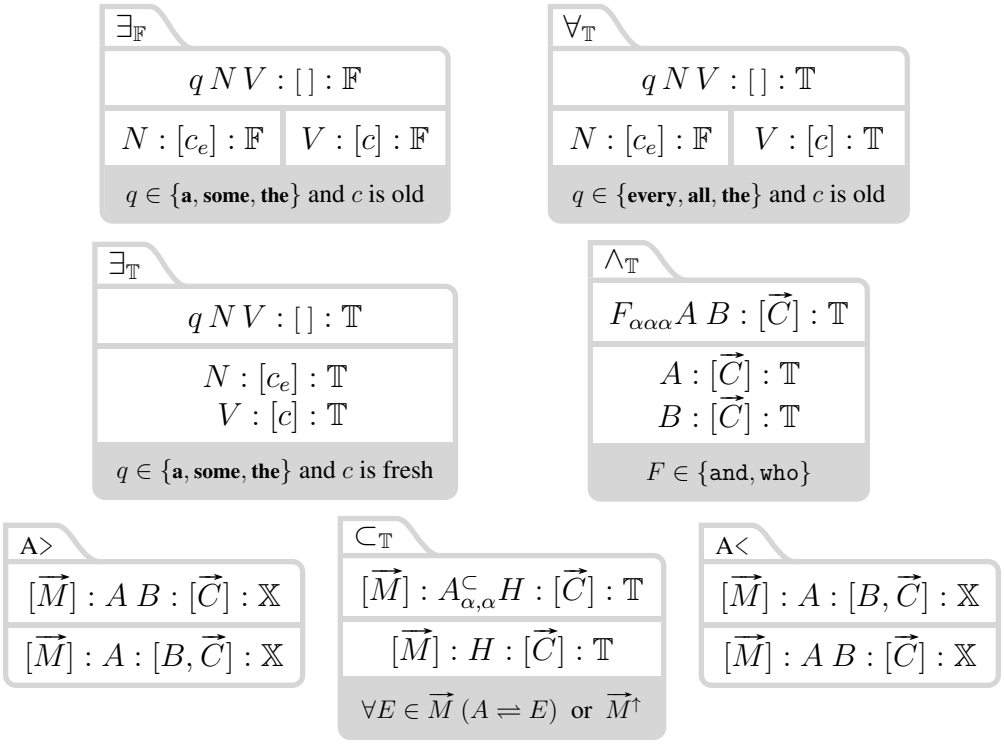


Figure 5.2: Some of the tableau rules

respectively.[6] ($\subset_{\mathbb{T}}$) and ($\exists_{\mathbb{F}}$) are examples of a semantic non-equivalence rule while (A>), (A<) and ($\wedge_{\mathbb{T}}$) are semantic equivalence rules. After applying a semantic non-equivalence rule to nodes in a tableau, the nodes have to be accessible for other rule applications; otherwise some semantic information of the nodes will be neglected on the branch possibly causing unsound results. From an application point of view, nodes should not be deleted from a branch when a semantic non-equivalence rule is applied to them.

Structural equivalence A tableau rule is *structural equivalence* with respect to the IR iff replacing its antecedents with its consequents does not change final status of any tableau proof; otherwise the rule is *structural non-equivalence*. Put differently, a rule is structural equivalence iff it is safe to ignore or discard the entries from the branch after the rule applies to them. For instance, (A<) and (A>) are structural equivalence rules (with respect to the IR) if both are in the IR. When exactly one of them is absent, then another one becomes structural non-equivalence. In particular, if there is only (A>) from them in the IR, then it can push all arguments of some entry in the argument list, make semantics of the entry inaccessible for other rules, and hence prevent the tableau from closing. Notice that a structural equivalence rule is automatically semantic equivalence and consequently a semantic non-equivalence rule is structural non-equivalence. Therefore, ($\subset_{\mathbb{T}}$) and ($\exists_{\mathbb{F}}$) are structural non-equivalence. Since structural equivalence is relative to the IR, it is a complex task to show whether a particular rule is structural equivalence or not. Due to

[6] While talking about antecedents and consequents in a collective way, we assume the Boolean connectives over them that are structurally encoded in tableau rules. For example, in case of a binary branching rule, a conjunction of its antecedents is contrasted to a disjunction of conjunctions of its left consequents and right consequents.

this reason, instead of the structural equivalence feature, we will use semantic equivalence in an efficiency feature vector.

Consuming A rule that can be triggered by the introduction of a fresh entity term on a branch is considered a *consumer*. For example, ($\exists_{\mathbb{F}}$) is a consumer rule as it employs already existing, i.e. old, entities on a branch. On the other hand, none of the other rules in Figure 5.2 are consumers, i.e. they are *non-consumers*. In general, consumer rules are not efficient from computational point of view—more entity terms lie on a branch more rule application options are there for a consumer rule. Note that a consumer rule is semantic non-equivalence automatically.

Producing Depending on whether a rule produces a fresh entity term, the rule is considered as a *producer* or a *non-producer*. Producer rules might induce significant inefficiency in conjunction with consumer rules. If we consider a tableau initiated with an LLF of a sentence "*everybody has somebody*" with the true sign where everybody has a wide scope, then the tableau will not terminate due to the infinite number of rule applications of consumer ($\forall_{\mathbb{T}}$) and producer ($\exists_{\mathbb{T}}$) rules that *feed* each other.

Some of the above mentioned properties have their proper names in the literature after Smullyan (1968). For instance, binary branching rules are called β-rules and non-branching ones α-rules. In first-order logic, the analogous rules to ($\exists_{\mathbb{F}}$) and ($\forall_{\mathbb{T}}$) are consumers and called γ-rules while the rule similar to ($\exists_{\mathbb{T}}$) is a producer and referred as a δ-rule.

For each above mentioned property, a tableau rule either has it or not. To determine a computational efficiency of a rule, we associate a binary 4-dimensional feature vector to each rule where dimensions stand for the features while the values 0 and 1 show attribution of these features to a rule:

$$\langle \texttt{nonBranching}, \texttt{semanticEquivalence}, \texttt{nonConsumer}, \texttt{nonProducer} \rangle$$

In this way, ($\exists_{\mathbb{F}}$) has an efficiency feature vector, in short *eff-vector*, $\langle 0, 0, 0, 1 \rangle$ since it is branching, semantic non-equivalence, consumer and non-producer. A set of eff-vectors represents a partition of tableau rules.

A componentwise order over the eff-vectors determines the *partial efficiency order* over the rules. Consequently, ($\wedge_{\mathbb{T}}$) with $\langle 1, 1, 1, 1 \rangle$ is more efficient than ($\subset_{\mathbb{T}}$) with $\langle 1, 0, 1, 1 \rangle$ while the latter is more efficient than ($\exists_{\mathbb{F}}$). On the other hand, ($\exists_{\mathbb{T}}$) with $\langle 1, 1, 1, 0 \rangle$ is not comparable to ($\exists_{\mathbb{F}}$) and ($\subset_{\mathbb{T}}$).

Later we show how the efficiency order over tableau rules can be used in a rule application strategy while building tableau proofs.[7]

5.2.2 Derivable rules

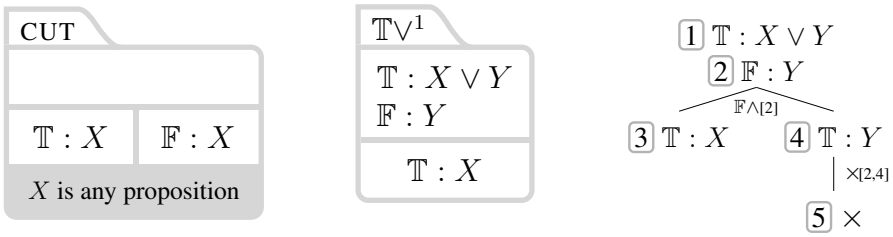
In order to decrease the size of automatically generated tableau proofs and make them more human-readable, in the computational model of the theorem prover, we add extra

[7] While talking about inefficiency of the tableau rules, we could also discuss inefficiency of tableau entries given the IR. For instance, **and** $A\,B : [\vec{C}] : \mathbb{T}$ can be seen as more efficient than **and** $A\,B : [\vec{C}] : \mathbb{F}$ based on how they are processed by the rules. But since this kind of efficiency property depends on the IR, which is a variable and dynamic component of the proof system, it will need update whenever the IR is modified. Moreover, we find it more natural to associate an efficiency measure to each tableau rule and control building of a tableau proof via selection of the rules.

derivable rules to the IR. These rules are redundant from the completeness point of view but their applications represent shortcuts for several rule applications. The derivable rules can also contribute to a rule application strategy as they are more efficient than the rules they emulate.

Large tableau proofs are unwanted for two main reasons. First, they are associated with time consuming computations and second, it is difficult for a human to analyze them. Unfortunately, it is not always possible to drastically decrease a search time for tableau proofs, but it is possible to make the proofs short by avoiding redundant rule applications. While potentially finding a short proof might require (slightly) more time than a long proof, it is still worthy to invest some effort in it as it significantly simplifies debugging and development of the theorem prover.

There are two notions of redundant inference rules. A rule is said to be *admissible* if its introduction does not yield any new theorem, i.e. for any initial tableau entries if the tableau does not close, it would not either close in case an admissible rule was adopted. If we consider the propositional tableau system from §2.0.2, then (CUT) is an admissible rule there. It is sound and does not contribute to the proof of any new theorem.[8]



A rule is said to be *derivable* if there exists a tableau that can mirror it. For example, ($\mathbb{T}\vee^1$) is derivable since there is a tableau (see on its right) that can simulate it. A derivable rule is automatically admissible as it simply represents a shortcut for several rule applications. On the other hand, not all admissible rules are derivable; for example, (CUT) is not a derivable rule. In the previous chapters, we already presented several derivable rules (with light gray background) for the natural tableau. In addition to them, we introduce new derivable rules below.

We start with the derivable rules related to (A>) and (A<). The latter rules are quite inefficient from a computational viewpoint; they can be infinitely applied after each other to a single tableau entry—the loop of pushing an argument and then pulling it back. To eliminate this dull sequence of rule applications, we replace (A<) with (λ<).[9] So, an argument is pulled from an argument list iff the corresponding LLF is formed from λ-abstraction.

Pushing arguments of any type with (A>) is often redundant. We restrict (A>) to (np>), where the latter pushes only the terms of type np (and e per se). Notice that after replacing (A<) with (λ<), the rules (A>) and (np>) become structural non-equivalence. But what about pushing the terms of type other than np? When certain constraints are met, the rules ($\uparrow\sqsubset$), ($\downarrow\sqsubset$), ($\nleftrightarrow$) and ($\nleftrightarrow$) from Chapter 2 push arguments of relational type in their

[8] The rule (CUT) is similar to a cut rule of sequent calculus. If a proposition P is derivable from a set of propositions Γ and Q is derivable from P and Δ, written as $\Gamma \vdash P$ and $\Delta, P \vdash Q$, then $\Gamma, \Delta \vdash Q$ holds with the help of (CUT). To show the latter, it is sufficient to apply (CUT) for P to a tableau started with $\Gamma \cup \Delta$.

[9] $A[x/C]$ is a term obtained from A by substituting x with C where C has no free occurrence of x.

right consequents. To allow more options for argument pushing, we introduce (A2>) which applies to the LLFs that share an argument (of any type) and pushes the arguments in the argument lists. Application of (A2>) can be seen as two applications of (A>) that plausibly leads a proof to a crucial point. In particular, it contrasts A and B while neglecting the shared argument C. Due to this property, the rule acts as an aligner for LLFs. Since aligning techniques are crucial for RTE systems (Dagan et al., 2013), we find (A2>) useful for the tableau theorem prover.

$\lambda<$

$$\frac{\lambda x.\, A : [C, \vec{D}] : \mathbb{X}}{A[x/C] : [\vec{D}] : \mathbb{X}}$$

np>

$$\frac{AC_{\mathtt{np}} : [\vec{D}] : \mathbb{X}}{A : [C, \vec{D}] : \mathbb{X}}$$

A2>

$$\frac{A\,C : [\vec{D}] : \mathbb{X} \quad B\,C : [\vec{D}] : \mathbb{Y}}{A : [C, \vec{D}] : \mathbb{X} \quad B : [C, \vec{D}] : \mathbb{Y}}$$

Half of the rules applicable to the existential and universal quantifiers are consumer rules, e.g., ($\exists_{\mathbb{F}}$) and ($\forall_{\mathbb{T}}$). To make application of these rules more efficient, we introduce two derivable rules for each of the rules. The derivable rules, given below, are incomplete but non-consumer versions of the quantifier rules. Their efficiency is due to choosing a relevant entity c_e rather than any entity, e.g., ($\forall^n_{\mathbb{T}}$) chooses the entity that satisfies the noun term while ($\forall^v_{\mathbb{T}}$) picks the one unsatisfying the verb term. Moreover, the derivable rules are not branching unlike their consumer counterparts. In the theorem prover, these six rules are implemented as three—incorporating a noun and a verb counterparts of the rules, e.g., ($\exists^n_{\mathbb{F}}$) and ($\exists^v_{\mathbb{F}}$), in one.

$\exists^n_{\mathbb{F}}$

$$\frac{q_{\mathtt{n,vp,s}}\,N\,V \;:\; [\,] : \mathbb{F} \quad N : [c_e] : \mathbb{T}}{V : [c] : \mathbb{F}} \qquad q \in \{\textbf{a}, \textbf{some}, \textbf{the}, \textbf{s}\}$$

$\forall^n_{\mathbb{T}}$

$$\frac{q_{\mathtt{n,vp,s}}\,N\,V \;:\; [\,] : \mathbb{T} \quad N : [c_e] : \mathbb{T}}{V : [c] : \mathbb{T}} \qquad q \in \{\textbf{every}, \textbf{the}\}$$

$\mathrm{NO}^n_{\mathbb{T}}$

$$\frac{\textbf{no}_{\mathtt{n,vp,s}}\,N\,V \;:\; [\,] : \mathbb{T} \quad N : [c_e] : \mathbb{T}}{V : [c] : \mathbb{F}}$$

$\exists^v_{\mathbb{F}}$

$$\frac{q_{\mathtt{n,vp,s}}\,N\,V \;:\; [\,] : \mathbb{F} \quad V : [c_e] : \mathbb{T}}{N : [c] : \mathbb{F}} \qquad q \in \{\textbf{a}, \textbf{some}, \textbf{the}, \textbf{s}\}$$

$\forall^v_{\mathbb{T}}$

$$\frac{q_{\mathtt{n,vp,s}}\,N\,V \;:\; [\,] : \mathbb{T} \quad V : [c_e] : \mathbb{F}}{N : [c] : \mathbb{F}} \qquad q \in \{\textbf{every}, \textbf{the}\}$$

$\mathrm{NO}^v_{\mathbb{T}}$

$$\frac{\textbf{no}_{\mathtt{n,vp,s}}\,N\,V \;:\; [\,] : \mathbb{T} \quad V : [c_e] : \mathbb{T}}{N : [c] : \mathbb{F}}$$

We have presented several derivable rules concerning the argument pushing and pulling rules and the certain rules for quantifiers. The derivable rules apply to more specific scenarios. Compared to the usage of the corresponding general rules, the application of the derivable rules results in shorter tableaux.

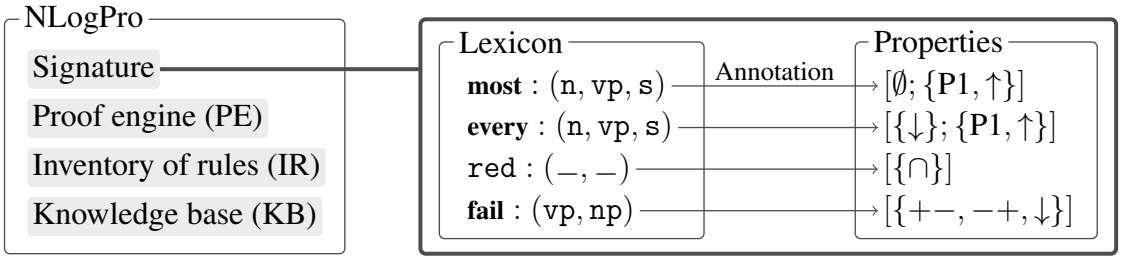


Figure 5.3: The architecture of NLogPro with the structure of the signature

5.3 NLogPro: a theorem prover for natural logic

Based on the natural tableau system, we implement a theorem prover for natural logic, called NLogPro. The prover can check a finite set of LLFs on consistency or prove theorems about LLFs via building a tableau for a finite set of LLFs. Below we describe the architecture of NLogPro (see Figure 5.3) and the rest of its components: the signature and the proof engine (PE). We also give the algorithm of the prover.[10]

NLogPro consists of four components: the signature, the PE, the IR and the KB, where the latter two have been already discussed in the previous subsections. These components collaborate as follows. Given a tableau branch, the PE finds a rule from the IR and entries on the branch such that the rule is applicable to the entries. While choosing a rule and entries, the PE appeals to the signature and the KB in order to check whether the terms in entries satisfy the constraints of a rule.

The signature represents a pair of a lexicon and the annotation function $\mathcal{A}$ (see §2.1). The lexicon represents a collection of lexical terms with the optional typing information (see Figure 5.3). The lexical terms in the lexicon are those terms that have certain algebraic properties. The annotation function maps each element of the lexicon to a list of sets of properties. For instance, **most** has no properties associated with its first argument while the second argument is upward monotone and satisfies the property (P1). **red** has an unspecified function type (e.g., it can be $(\mathtt{n}, \mathtt{n})$ or (et)) and its first argument has the intersective property (and upward monotonicity property automatically, but the latter is not necessary to be explicated as the rule for intersective adjectives accounts for the monotonicity property too). The implicative verb **fail** has three properties associated with the first argument and no properties to its second argument. By convention we omit an empty property set for trailing argument positions.

The PE is a component responsible for construction of tableau proofs; it also maintains a rule application strategy. Given a tableau, the PE finds a rule that will be applied to the entries of the tableau next. To make the rule application strategy deterministic it is sufficient to have a *linear priority order* ($<_{pr}$) over tableau rules: if there are several rule applications possible, apply the most prior rule among them. A notion of efficiency in the PE is introduced by an *efficiency criterion* which represents a linear priority order over the components of an eff-vector. An efficiency criterion EC induces a linear alphabetical

[10]The theorem prover is implemented in Prolog. Initially it started as a first-order logic (FOL) prover following Fitting (1990), but later its was substantially extended for the simple type theory. The prover also contains a module for the λ-calculus which roughly follows Blackburn and Bos (2005).

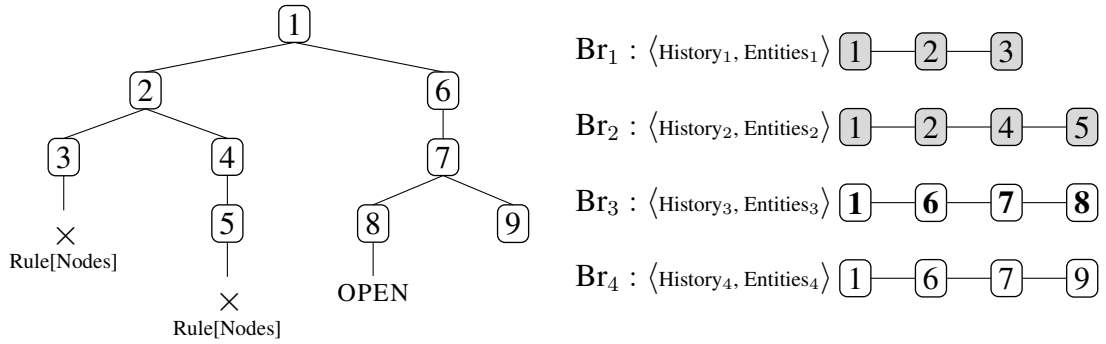


Figure 5.4: A tree and a list structure developed by the PE

order ($<_{EC}$) over eff-vectors. For instance, the criterion

$$EC = [\texttt{nonProducer}, \texttt{nonBranching}, \texttt{semanticEquivalence}, \texttt{nonConsumer}]$$

where the features are sorted in priority descending, induces the following linear order over the eff-vectors of the rules discussed in §5.2.1:

$$(\exists_{\mathbb{T}})_{\langle 0,1,1,1\rangle} <_{EC} (\exists_{\mathbb{F}})_{\langle 1,0,0,0\rangle} <_{EC} (\subset_{\mathbb{T}})_{\langle 1,1,0,1\rangle} <_{EC} (\wedge_{\mathbb{T}})_{\langle 1,1,1,1\rangle}$$

where each rule is accompanied by its eff-vector (modified according to EC):

Definition 17 (Linear efficiency order)**.** Given a linear priority order ($<_{pr}$) and an alphabetical order ($<_{EC}$) induced by some efficiency criterion EC, for any r_1 and r_2 tableau rules, a *linear efficiency order* ($<_{\mathit{eff}}$) is defined as:

(a) $r_1 <_{\mathit{eff}} r_2$ if $\texttt{effvec}(r_1) <_{EC} \texttt{effvec}(r_2)$;

(b) $r_1 <_{\mathit{eff}} r_2$ if $\texttt{effvec}(r_1) =_{EC} \texttt{effvec}(r_2)$ and $r_1 <_{pr} r_2$;

where $\texttt{effvec}(r)$ is the eff-vector of r. We read $r_1 <_{\mathit{eff}} r_2$ as r_2 is more efficient than r_1.

In other words, when deciding which rule is more efficient, we first appeal to the corresponding eff-vectors and ($<_{EC}$). If the eff-vectors are the same, i.e. the rules are in the same efficiency class, then ($<_{pr}$) is used. Notice that ($<_{\mathit{eff}}$) could also be trivially defined as ($<_{pr}$). The PE employs ($<_{\mathit{eff}}$) to decide which rule application to carry out next. Defining ($<_{\mathit{eff}}$) in terms of ($<_{pr}$) and ($<_{EC}$) has the following advantages: ($<_{EC}$) facilitates search for an optimal efficiency criterion and ($<_{pr}$) allows to prioritize rules according to the phenomena they account for.

We represent natural tableau proofs as upside-down trees because a tree structure shows a development process of a proof in a compact way. From a theorem proving perspective, a tree structure is inefficient data structure: accessing a single branch is not trivial and maintaining all tableau nodes during a proof procedure is not necessary. More suitable data structure for theorem proving is a list of branches (Fitting, 1990), but it is difficult to read the development process of a proof from the list. Due to these reasons, the PE simultaneously builds a tree and a list structure for each tableau proof (see Figure 5.4). A list structure is used in theorem proving and is maintained in a destructive way—closed branches and antecedents of equivalence rules are discarded. A tree structure is built in a constructive way according to the rule applications found on the list structure and it is

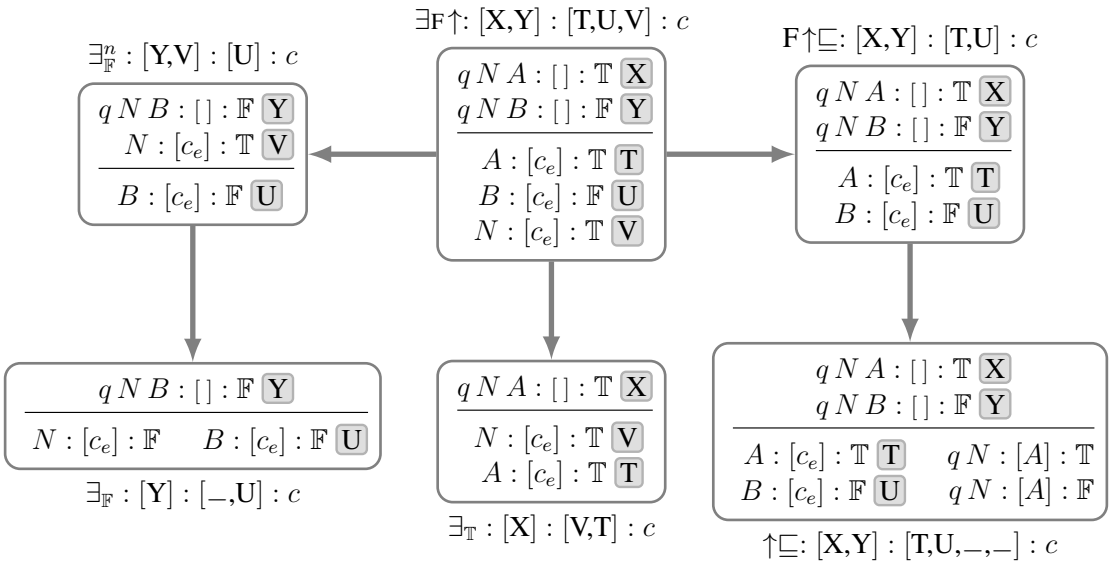


Figure 5.5: The specificity relation (⇒) over rule applications depicted visually, where arrows are directed from more specific rule applications to more general ones. The IDs of nodes that are irrelevant for the given relations are omitted.

intended for analysis of proofs. For each step, first a list structure is developed by finding a rule and applying it to entries and then a tree structure is expanded according to the rule application.

In the list structure, in addition to entries on a branch, the PE also tracks down a rule application history and maintains a set of entities for each branch (see Figure 5.4). A branch is interpreted as a situation with its own set of entities—the entity terms that occur in the entries of the branch. Hence, keeping a set of entities for each branch makes rule application of consumer rules easy and fast. On the other hand, there are semantically or structurally non-equivalence rules in the IR, and they can be infinitely applied to the same entries if no history of rule applications is recorded and no duplicate rule applications is forbidden.

Maintaining a history for each branch is also important from the perspective of derivable rules. For instance, if ($\exists^n_{\mathbb{F}}$) is applied to the nodes [Y] and [V] and introduces a node [U], then it is unnecessary to apply ($\exists_{\mathbb{F}}$) to [Y] and the entity in [V] (see Figure 5.5). This information is encoded in terms of a *specificity* relation (⇒) over rule applications, where a rule application is a tuple consisting of a rule, the IDs of antecedent and consequent nodes, and information about entities:

$$\exists^n_{\mathbb{F}} : [\text{Y,V}] : [\text{U}] : c \Rightarrow \exists_{\mathbb{F}} : [\text{Y}] : [\_,\text{U}] : c$$

We read the latter as the application of ($\exists^n_{\mathbb{F}}$) is more *specific* than of ($\exists_{\mathbb{F}}$), i.e. ($\exists_{\mathbb{F}}$) is more *general* than ($\exists^n_{\mathbb{F}}$). The relation (⇒) is a partial order over rule applications. For instance, in Figure 5.5, it is shown how the application of ($\exists$F↑) makes other five rules applications redundant (two of them in a transitive way). The specificity relation is defined in the IR. After the PE carries out a rule application on a tableau branch, all rule applications that are more general than the performed one are also added in the history of the branch. Hence, later the general rule applications will be banned by the PE.

In order to make sure that each branch gets its fair share of rule applications, after the PE carries out a rule application on a branch, it processes the next branch; in case of the last branch, the first branch is processed. In average, this strategy guarantees finding an open branch, if it exists, earlier compared to the strategy where a shift to a new branch happens only if the current one closes.

We do not want to wait indefinitely for termination of the prover. Hence, we set a *rule application limit* (RAL) serving as an upper bound for the number of rule applications in a single tableau. As a result, when the prover terminates, there are three outcomes available: all branches are closed, there is an open branch that cannot be further grown and there is an open branch but RAL is reached. From these three options, only in the first case we can say that a proof is found.

The pseudocode of NLogPro is presented in algorithm 2 (Appendix D). Given a list of signed tableau entries, a RAL number n and an efficiency criterion, NLogPro checks the list of entries on consistency, i.e. finds a situation that satisfies the entries. The prover starts with a list structure which contains a single branch involving all input tableau nodes labeled with IDs (see line 1 in algorithm 2). The local signature of the branch contains constant terms of type e or np occurring in the input nodes. The history of the branch is empty in the beginning. To define the efficiency order ($<_{\mathit{eff}}$), first the rules irrelevant to the input nodes are filtered out of the IR, and then based on the efficiency criterion the remaining rules are sorted in descending efficiency. The prover builds both the tree and the list stricture simultaneously using at most n number of steps. The predicate FINDRULE/3 in line 2 succeeds if r is the most efficient rule applicable to the nodes of a branch. After the rule is found it is applied to the corresponding nodes in the list and the tree structure. While applying the rule to nodes, the signature and the history of the branch is updated accordingly—fresh entities, if any, are added to the signature and the history is augmented with the rule application and those that are more general than the latter. Closed branches are deleted from the list and therefore an empty list amounts to a closed tableau.

We have presented NLogPro which checks signed LLFs on consistency. Put differently, it serves as a tableau proof builder. While building a tableau, the prover takes into account the linear efficiency order ($<_{\mathit{eff}}$) over the rules and applies the most efficient rule first. NLogPro builds a tree and a list structure in parallel: the list structure guides the proof procedure and the tree structure is used for analyzing proofs. Due to an ample amount of derivable and semantically non-equivalence rules, a history of rule applications is recorded for each branch. The latter also takes into account the specificity relation ($\Rightarrow$) over rule applications in order to avoid redundant rule applications. For more details we have also presented the pseudocode of NLogPro. In the next subsection, we will describe how NLogPro is integrated in an RTE system.

5.4 LangPro: a theorem prover for natural language

Up to now we have described all the tools that are necessary to obtain a theorem prover for natural language based on the natural tableau. In particular, by chaining a CCG parser, LLFgen (Chapter 3) and NLogPro (§5.3), we get a natural language theorem prove, called LangPro. Below we describe the architecture (Figure 5.6) and functionality (Figure 5.7 and algorithm 3) of LangPro.

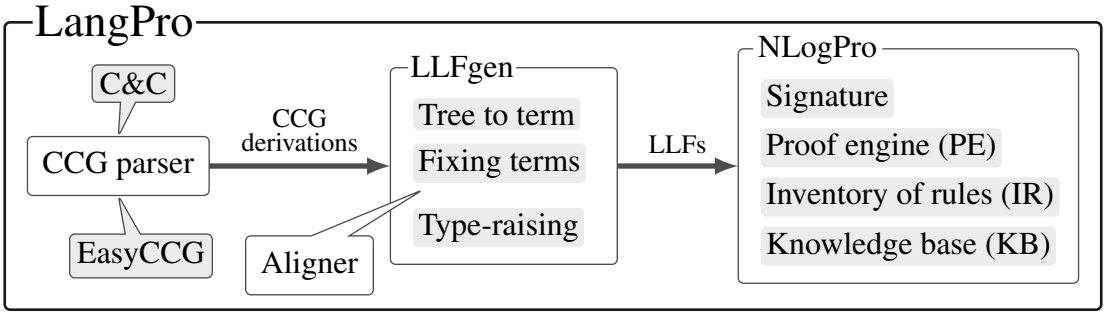


Figure 5.6: The architecture of LangPro

LangPro represents a tableau-based theorem prover for natural language. It takes a set of premises and a conclusion in English and returns one of the logical relations—entailment, contradiction or neutral—between their semantics. Input linguistic expressions are not limited to any specific syntactic category but they must be of the same category. For instance, it can reason over noun, verb phrase or even over prepositional phrases. If a parser is unable to parse one of the linguistic expressions or does not assign the same syntactic category to each of the expressions, then LangPro does not reason further; it reports the issue and classifies the input problem as neutral (see line 1 in algorithm 3, Appendix D).

LangPro has a pipeline architecture (see Figure 5.6). Its input expressions are first parsed by a CCG parser, either C&C (Clark and Curran, 2007) or EasyCCG (Lewis and Steedman, 2014a). The CCG derivations are then processed by LLFgen which generates a list of LLFs for each derivation differing in a scope order of quantifiers. In the prover, LLFgen is augmented with an aligner component that aligns common chunks of terms (discussed later). Finally, the obtained LLFs are fed into NLogPro where the latter builds several tableaux to identify a logical relation between the premises and the conclusion. These procedures are demonstrated on a concrete entailment problem in Figure 5.7.

Often premises and a conclusion share several multiword constituents analysis of which is not relevant for the correct classification of the problem. Due to this reason, many RTE systems adopt alignment techniques that help the systems to concentrate on relevant parts of text (Dagan et al., 2013). In LangPro alignment is carried out by the term aligner—an optional component. It grabs all fixed CCG terms from LLFgen, finds compound terms shared by all the terms and replaces them with fresh constant terms. While doing so, the aligner makes sure that shared terms are not downward monotone; see the aligner in action in Figure 5.7. The corresponding aligned LLFs are then obtained by type-raising the aligned fixed CCG terms. We choose to align fixed CCG terms over type-raised ones because the former terms are closer to surface forms rather than the latter ones. A tableau initiated with aligned LLFs is shorter than the one with original, i.e. non-aligned, LLFs. Therefore, employing aligned LLFs increases the chance of finding a proof in the limited number of rule applications. However, committing only to aligned LLFs might eliminate a chance of finding a proof—the semantics of shared constituents are not accessible in a tableau. For this reason, LangPro feeds original LLFs into NLogPro after the latter is not able to find a proof with the aligned LLFs (see line 2 in algorithm 3). This implies that in the worst case, e.g. for textual entailments with the neutral relation, NLogPro ends up with building four tableaux (see Figure 5.7).

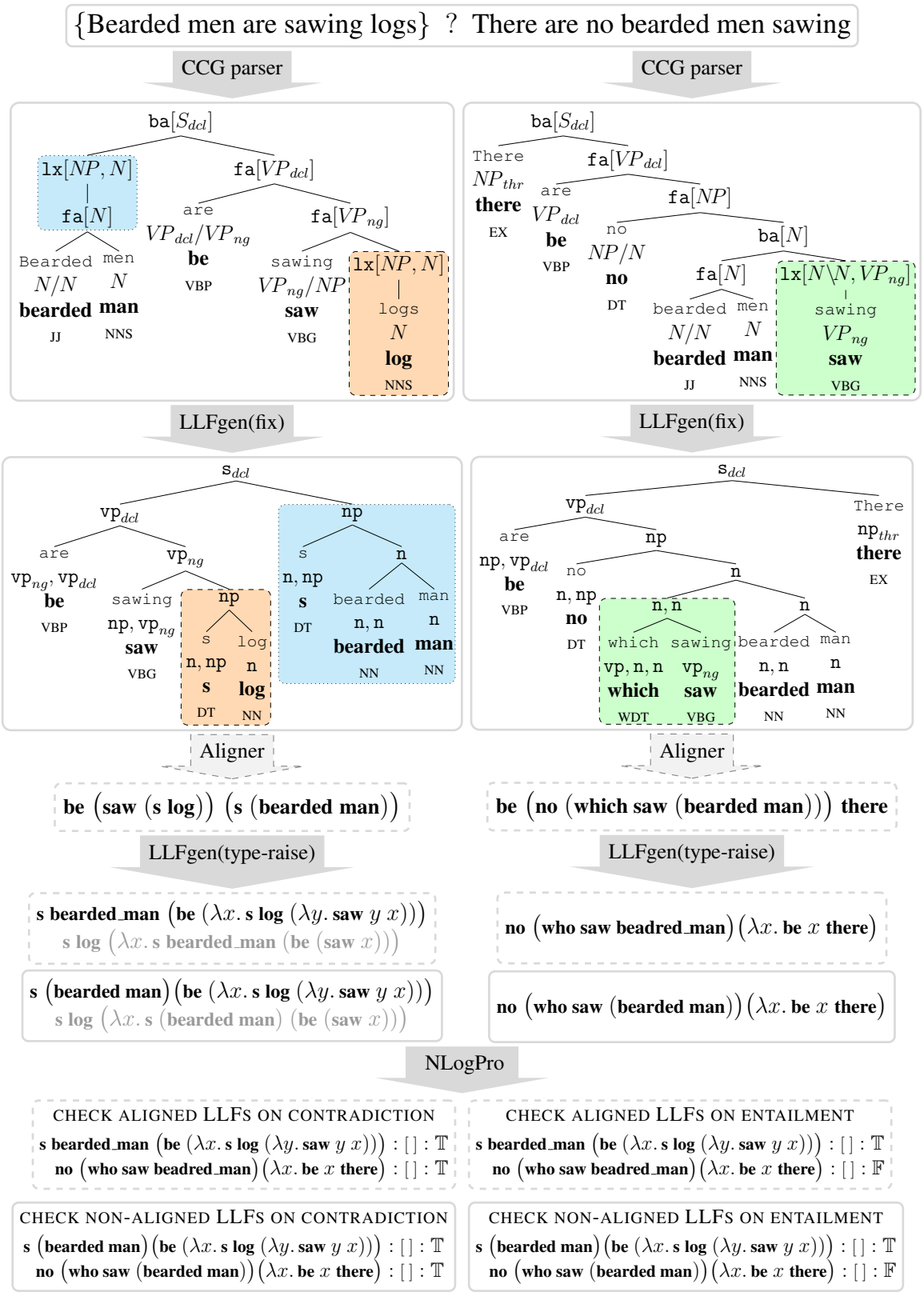


Figure 5.7: Reasoning by LangPro: in CCG derivations and trees, constituents with the type changing rule `lx` and their fixed versions are framed; the optional aligner component and its outputs are in dashed boxes. NLogPro is fed with the first type-raised LLFs.

In order to decrease the number of tableaux per entailment problem, for each linguistic expression LangPro employs a single CCG derivation[11] and a single LLF (modulo alignment), demonstrated in Figure 5.7. The prover adopts the most probable CCG derivation for each expression. Another type of indeterminacy comes from the type-raising procedure of LLFgen when it returns a list of LLFs due to different scope orders of quantifiers. LangPro employs the very first LLF from a list since its scope order is closest to the order occurring in a surface form; see Figure 5.7 for an example. For the same reason—to decrease the number of tableaux—CCG derivations from different parsers are not coupled. Moreover, we do not expect a significant improvement from the latter approach as compared to the approaches based on a single parser there is a higher chance that the same constituents are parsed differently, i.e. parsing is less consistent or deterministic.

Closure rules usually require access to the KB. For efficient theorem proving, LangPro redefines a small KB for NLogPro relevant to an input entailment problem. It first extracts lexical elements from fixed CCG terms and then based on the WordNet relations defines the semantic relations between the elements (see §5.1). This step prevents LangPro from accessing WordNet more than once for each input problem and from considering irrelevant lexical entries. Further details about the functionality of LangPro can be found in algorithm 3 (Appendix D).

We have described the natural language prover LangPro by outlining its architecture, described in details how it processes a sample entailment problem in Figure 5.7 and give its pseudocode in algorithm 3. An online demo of LangPro is available at `http://www.naturallogic.pro/langpro/`

5.5 Conclusion

In this chapter we have presented implementation details of the natural tableau-based theorem prover. We started with the KB settings and explained how WordNet relations can be imported in the KB without word sense disambiguation. Based on the four efficiency properties of tableau rules (i.e. branching, semantic equivalence, consuming and producing), we have defined an efficiency feature vector (eff-vector) for each tableau rule. An eff-vector is supposed to model computational efficiency of the rule. Given a linear priority order over the efficiency properties (i.e. an eff-criterion), the rules can be partially ordered according to their eff-vectors. The partial order can be linearized with some default priority order over the rules. The linearized version of the order is used in the implemented natural logic prover (NLogPro) to identify applicable rules. As we will see later in §6.2.2, an optimal eff-criteria boosts the accuracy of the theorem prover on certain RTE datasets.

NLogPro employs a sophisticated procedure to develop a tableau tree. It builds both a tree and a list structures. The tree structure is easy to read while the list structure is only intended to be used for finding a next applicable rule. The prover records rule applications to make sure that no rule applies to the same entries two times. A set of derivable rules are mainly introduced to decrease tableau proofs in size as shorter proofs are easy to read. Due to the derivable rules, the specificity relation is maintained in order

[11] EasyCCG is able to return multiple CCG derivations for a single input while C&C incorporates only multi suppertagger—returning several CCG categories for each lexical entry.

to prevent redundant rule applications. Combining a CCG parser, the LLF generator, and NLogPro, we obtain the tableau-based theorem prover for natural language, called LangPro. In addition to this obvious pipeline, we incorporate an optional aligner into the LLF generator. The aligner searches and aligns shared sub-terms in corrected CCG terms. Aligned sub-terms are then treated as constant terms by NLogPro. As we will show in §6.2.2, the aligner significantly increases the performance of the theorem prover.

The architecture of LangPro is a simple pipeline consisting of the WordNet-based KB, one of the CCG parsers, the LLF generator and the prover NLogPro. So, the prover is purely compositional and logic-based. One of our goals is to check how well such logic-based RTE system scales up when employed natural logic-style reasoning.

In the next chapter, we will further develop the theorem prover on a training data. In particular, we will detect optimal values of several parameters such as the number of word senses in consideration, an efficiency criterion for rules, settings of the aligner and an upper bound of rule applications. After the optimal parameters are set, LangPro will be evaluated against the test data and the results will be compared to related RTE systems.

Appendix D

Algorithm 2: Pseudocode of the natural logic prover NLogPro. The prover checks signed nodes on consistency in a limited number of rule applications.

Input: A list of signed nodes *Nodes*, a rule application limit n and an efficiency criterion *EC*

Output: *Status* of a tableau: either `"closed"`, `"limited"` or `"open"`

Sig $\leftarrow$ `ExtractSignature`(*Nodes*) `// Extract entities from initial nodes`

Nids $\leftarrow$ `AddIDs`(*Nodes*) `// Label nodes with IDs`

1 *List* $\leftarrow$ [[ ] : *Sig* : *Nids*] `// List initially is a single branch with an empty history`

Tree $\leftarrow$ `CreateTree`(*Nids*) `// Create the corresponding tree structure`

Rules $\leftarrow$ `RelevantRules`(*Nodes*) `// Filter out rules irrelevant to the nodes`

Rules $\leftarrow$ `EfficientOrd`(*EC*, *Rules*) `// Sort rules in descending efficiency`

Steps $\leftarrow 0$

$i \leftarrow 1$ `// The first branch has an index 1`

while *Steps* $\leq n$ **do**

2 **if** FINDRULE(*List*[i], *Rules*, *IDs*, r) **then** `// Find a rule r which is applicable to the nodes with the labels IDs`

List[i] $\leftarrow$ `ApplyRule`(r, *List*[i], *IDs*)

GROWTREE(*Tree*, *IDs*, r) `// Develop Tree according to the rule application`

Steps $\leftarrow$ *Steps* $+ 1$

if CLOSED(*List*[i]) **then**

DELETEELEMENT(*List*, i) `// Discard a closed branch from List`

if *List* $= [\,]$ **then** `// If all branches are closed`

return `"closed"` `// Returns values and halts`

else

$i \leftarrow i$ `mod` $|$*List*$|$ `// Choose the next branch`

else

$i \leftarrow (i + 1)$ `mod` $|$*List*$|$ `// Choose the next branch`

else

return `"open"`

return `"limited"`

Algorithm 3: Pseudocode of the natural language prover LangPro. Given a list of premises and a conclusion, the prover detects a logical relation between them.

Input: A list of premises $P = [p_i]_1^n$ and a hypothesis h in English
Output: One of the relations `"ent"`, `"cont"` and `"neut"`

```
1 if (Trees ← map(CCGparse, P ⊕ [h]) and SameCat(Trees)) then
      // If all the expressions are parsed with the same syntactic category
      Terms ← map(Der2Term, Trees)    // Convert CCG derivations into CCG terms
      Fixed ← map(FixTerm, Terms)                    // Fix errors in CCG terms
      AFixed ← Align(Fixed)                           // Align fixed CCG terms
      LLFs ← TypeRaise(Fixed)[1]      // Pick the first LLF with type-raise GQs
      ALLFs ← TypeRaise(AFixed)[1]
      Lex ← GetLexicon(Fixed)          // Induce a lexicon from fixed CCG terms
      SetKBFromLexWN(Lex)        // Set a KB consisting of semantic relations
      between the elements of the lexicon defined in terms of the WordNet relations
      Align ← Solve(ALLFs)                    // Solve is separately defined below
2     if Align ≠ "neut" then
          return Align   // Adopt a proof of a non-neutral relation over aligned LLFs
      else
          return Solve(LLFs)                // Otherwise classify non-aligned LLFs
  else
      raise "Error in parsing"    // Raise an exception for a mistake in parsing
      return "neutral"


// Function Solve is a three-way classifier of textual entailment problems
Function Solve(P : list of LLFs, h : LLF) : String("ent","cont","neut") is
      TrueP ← map(AddTrueSign, P)
      EntNodes ← TrueP ⊕ [h : 𝔽]                          // ⊕ concatenates lists
      ContNodes ← TrueP ⊕ [h : 𝕋]
      switch ⟨NLogPro(EntNodes), NLogPro(ContNodes)⟩ do
          // For simplicity we assume that a rule application limit and an efficiency
          criterion are already implicitly set in NLogPro (see algorithm 2)
          case ⟨"closed","closed"⟩ do  // Usually indicates mistakes of a parser
              raise "Entailment & contradiction"
              return "neut"
          case ⟨"closed","open"⟩ do
              return "ent"
          case ⟨"open","closed"⟩ do
              return "cont"
          otherwise do
              return "neut"
```

Chapter 6

Evaluation of the theorem prover

We have presented the generator for LLFs and the inventory of tableau rules in Chapter 3 and Chapter 4 respectively. The natural language theorem prover that incorporates both of these components has been described in Chapter 5. In fact, the development of the prover and its internal components were done simultaneously. While we were increasing the number of rules in the inventory, at the same time, we kept improving the LLF generator in terms of the fixing rules. In this chapter we are going to describe this development phase in details and then evaluate the *mature*, developed, prover on empirical RTE datasets. For the development and evaluation, we use two datasets FraCaS and SICK consisting of problems different in nature. While the former concentrates on deep reasoning over multiple-premised problems, the latter incorporates single-premised problems requiring relatively shallow reasoning. After we describe these two datasets and present sample problems from them, we switch to a data-driven learning phase. The phase represents an umbrella term for what NLP community call adaptation and development. During the adaptation, the tableau prover is upgraded in order to solve some entailment problems from a dataset. The upgrade involves introducing either a new tableau rule, a new fixing rule for the generator, a new entry in the signature or a lacking lexical relation. But it is not always possible to solve a problem in the adaptation. After the adaptation procedure, the best configuration of the prover is defined in the development procedure. The configuration concerns an approach to word senses, the aligner component, an efficiency criterion and the rule application limit. We also analyze the results of the development procedure. The developed prover is evaluated on the corresponding datasets. Obtained results are analyzed and compared to the related RTE systems. We also contrast our prover and its underlying theory with other systems and their approaches in a qualitative way. To the best of our knowledge, our theorem prover is the only system that is evaluated on both the FraCas and SICK datasets.

6.1 RTE datasets

We need specially prepared RTE datasets in order to develop and evaluate the theorem prover. We choose the Fracas and the SICK datasets for this purpose. The datasets are chosen because of their relatively short sentences. We expect the CCG parsers to make fewer mistakes on the short sentences and provide the theorem prover with quality derivations. On the other hand, the datasets concentrate on different phenomena. Composi-

tionality and lexical knowledge are crucial for the SICK problems while FraCaS covers semantically challenging phenomena like generalized quantifiers, plurals and attitudes encoded in multi-premised problems. By employing the datasets, we intend to train and evaluate the prover for both deep and shallow reasoning.

6.1.1 SICK

SICK (Sentences Involving Compositional Knowledge) by Marelli et al. (2014b) is an RTE dataset which is intended as a benchmark for Compositional Distributional Semantic Modelss (CDSMs). The data contains the lexical, syntactic and semantic phenomena that CDSMs are expected to account for; some of these phenomena are contextual synonymy, active/passive alternations, negation and quantifiers. In general, solving the SICK problems does not require recognizing named entities, identifying multiword expressions, or having encyclopedic knowledge. SICK contains about 10,000 English sentence pairs each annotated with two gold answers using crowdsourcing. First is a score (from 1 to 5 points) that measures relatedness of the meanings of sentences. Second is a three-way label—entailment, contradiction or neutral—encoding the semantic relation from the first sentence to the second one. Several SICK sentence pairs are given in Table 6.2.

Original pair	
S0a: *A sea turtle is hunting for fish*	**S0b:** *The turtle followed the fish*
Normalized pair	
S1a: *A sea turtle is hunting for fish*	**S1b:** *The turtle is following the fish*
Expanded pair	
Similar meaning	
S2a: *A sea turtle is hunting for food*	**S2b:** *The turtle is following the red fish*
Logically contradictory or at least highly contrasting meaning	
S3a: *A sea turtle is not hunting for fish*	**S3b:** *The turtle isn't following the fish*
Lexically similar but different meaning	
S4a: *A fish is hunting for a turtle in the sea*	**S4b:** *The fish is following the turtle*

Table 6.1: Normalized sentences produced from an original pair

The SICK sentence pairs are produced semi-automatically from descriptions of pictures and videos and are annotated by humans. In Table 6.1, there is a real example that shows how eight sentences **S1**-**S4/a**-**b** are obtained from a single pair **S0a**-**S0b** of captions describing the same picture or video. The final entailment pairs are produced by pairing all the expanded sentences with the (source) normalized ones (hence, forming 12 problems) and pairing the normalized ones with each other too. The resulted 13 pairs are then annotated using crowdsourcing, where a relatedness gold score is an average of assigned scores and an entailment gold label represents a major assigned label. These 13 pairs with the gold scores and labels are given in Table 6.2.

The distribution of gold labels and relatedness (i.e. similarity) scores in the SICK data is given in Table 6.3. The majority of the problems ($\approx$57%) are labeled as neutral. This is mainly conditioned by cross pairs (**S**X**a**, **S**X**b**) being logically neutral to each other. The relation between relatedness scores and the entailment label is clearer than in case of the neutral and contradiction labels. The reason is that the neutrally labeled problems

Normalized sentence pairs		Score	Label
S1a: *A sea turtle is hunting for fish*	**S2a:** *A sea turtle is hunting for food*	4.5	E
S3a: *A sea turtle is not hunting for fish*	**S1a:** *A sea turtle is hunting for fish*	3.4	C
S4a: *A fish is hunting for a turtle in the sea*	**S1a:** *A sea turtle is hunting for fish*	3.9	N
S2b: *The turtle is following the red fish*	**S1b:** *The turtle is following the fish*	4.6	E
S1b: *The turtle is following the fish*	**S3b:** *The turtle isn't following the fish*	4	C
S1b: *The turtle is following the fish*	**S4b:** *The fish is following the turtle*	3.8	C
S1a: *A sea turtle is hunting for fish*	**S2b:** *The turtle is following the red fish*	4	N
S1a: *A sea turtle is hunting for fish*	**S3b:** *The turtle isn't following the fish*	3.2	N
S4b: *The fish is following the turtle*	**S1a:** *A sea turtle is hunting for fish*	3.2	N
S1b: *The turtle is following the fish*	**S2a:** *A sea turtle is hunting for food*	3.9	N
S1b: *The turtle is following the fish*	**S3a:** *A sea turtle is not hunting for fish*	3.4	N
S4a: *A fish is hunting for a turtle in the sea*	**S1b:** *The turtle is following the fish*	3.5	N
S1a: *A sea turtle is hunting for fish*	**S1b:** *The turtle is following the fish*	3.8	N

Table 6.2: SICK entailment pairs with human annotation

have a quite wide range of relatedness scores and, at the same time, most of neutral and contradiction problems have scores in $[3, 4)$ interval. These facts make the prediction from scores to the contradiction and neutral problems obscure.

Relatedness	NEUTRAL	CONTRADICTION	ENTAILMENT	Total
[1,2) range	10%	0%	0%	10% (923)
[2,3) range	13%	1%	0%	14% (1373)
[3,4) range	28%	10%	1%	29% (3872)
[4,5] range	7%	3%	27%	37% (3672)
Total	56.86% (5595)	14.47% (1424)	28.67% (2821)	9840

Table 6.3: Distribution of gold labels and relatedness scores in SICK

The SICK dataset was used as a benchmark for the shared tasks in semantic relatedness and textual entailment at SemEval-14 (Marelli et al., 2014a). For the tasks, the data was partitioned into three parts: SICK-trial (500), SICK-train (4500) and SICK-test (4927).[1] Hereafter we employ the version of SICK adopted for SemEval-14.[2]

6.1.2 FraCaS

The FraCaS semantic test suite (Cooper et al., 1996) was carefully designed by members of the FraCaS consortium in order to create an initial version of a collection of inference tasks that would serve as the best way for evaluating semantic capacity of NLP systems. An inference problem in the test suite consists of a set of premises, a single yes-no question and a gold answer, which is most of the time either YES, NO or DON'T KNOW; for example, an exception is FraCaS-12 in Figure 6.1. Sometimes a problem comes with a comment that explains the answer, e.g., see FraCaS-87, 88. The entailment labels and comments are provided by the members of the FraCaS consortium (Cooper et al., 1996).

[1]Notice that the SICK data used for the SemEval challenges (Marelli et al., 2014a) contains slightly more problems than the data described in Marelli et al. (2014b) and Table 6.3.

[2]The partitioned dataset is available from http://alt.qcri.org/semeval2014/task1/index.php?id=data-and-tools

FraCaS-12 *Generalized quantifiers*:
Few great tenors are poor.

Are there great tenors who are poor?
[Not many]

FraCaS-33 *Generalized quantifiers*:
An Irishman won a Nobel prize.

Did an Irishman win the Nobel prize for literature?
[DON'T KNOW]

FraCaS-49 *Generalized quantifiers*:
A Swede won a Nobel prize.
Every Swede is a Scandinavian.

Dis a Scandinavian win a Nobel prize?
[YES]

FraCaS-87 *Plurals*:
Every representative and client was at the meeting.

Was every representative at the meeting?
[YES, on one reading]
Arguably NP conjunction: every representative and every client

FraCaS-258 *Temporal reference*:
In March 1993 APCOM founded ITEL.

Did ITEL exist in 1992?
[NO]

FraCaS-88 *Plurals*:
Every representative and client was at the meeting.

Was every representative at the meeting?
[DON'T KNOW, on one reading]
NBar conjunction: everyone who is both a representative and a client

Figure 6.1: Original problems from the FraCaS test suite

The FraCaS problems, on one hand, resemble tasks found in introductory logic books, where it is asked to derive a given test sentence as a conclusion from a small number of premises or to construct a counterexample. On the other hand, they show similarity to problems for *text analysis* that consists of a text and a yes-no question related to it. The test suite contains 346 inference problems where 45% of them have multiple premises; on average there comes 1.55 premises per problem. The problems are classified in 9 sections, where each section concentrates on a specific semantic phenomenon: generalized quantifiers, plurals, (nominal) anaphora, ellipsis, adjectives, comparatives, temporal reference, verbs and attitudes (see Table 6.4).

The test suite became popular in NLP community after it was converted in the format of RTE data and used for evaluation of an RTE system in (MacCartney and Manning, 2007). During the conversion procedure a hypothesis, which is a declarative equivalent of a yes-no question, is added to each problem. Also the answers are mapped to canonical values: YES, NO, UNKNOWN and UNDEFINED, where the first three corresponds to ENTAILMENT, CONTRADICTION and NEUTRAL respectively. Hereafter, we refer the converted version of the test suite as the FraCaS RTE data or simply the FraCaS data. Figure 6.2 represents the textual entailment problems from the FraCaS data corresponding to those problems of Figure 6.1. Statistics of each FraCaS section are given in Table 6.4.[3]

Compared to other RTE data sets, the FraCaS data is a very small in size, especially, when taking into account the diversity of phenomena it covers. For instance, phenomena like adjectives, verbs and attitudes are represented by only 44 problems. Moreover, the dataset contains short sentences and some of them even ambiguous. Due to ambiguous sentences, there are some identical RTE problems with different answers, like FraCaS-87,

[3]More details about the conversion, including information about several *noisy* problems (e.g., a problem missing a premise or hypothesis, or having a non-standard gold answer) can be found in (MacCartney, 2009). The FraCaS RTE dataset is available at: http://www-nlp.stanford.edu/~wcmac/downloads/fracas.xml

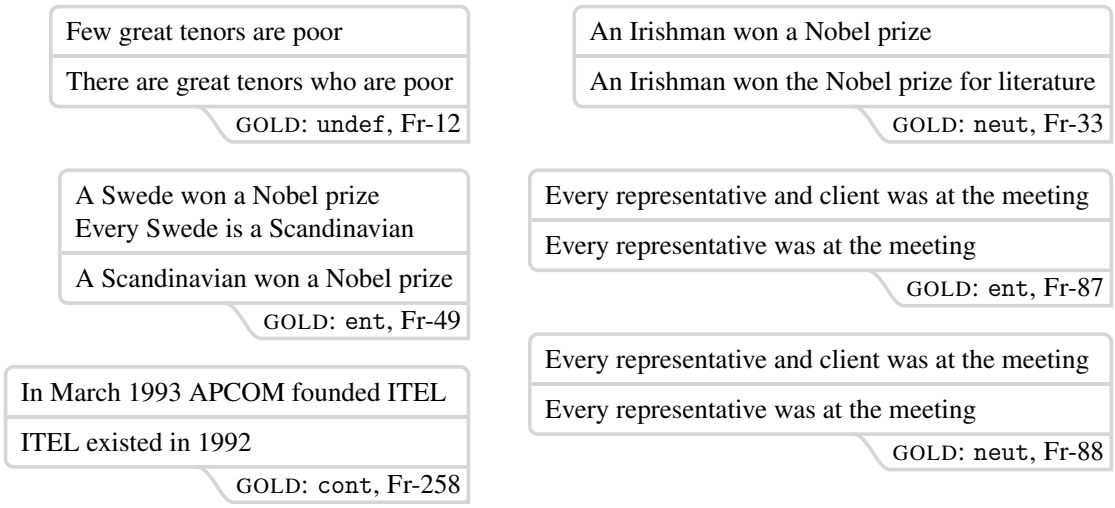


Figure 6.2: Textual entailment problems from the obtained FraCaS RTE data

88.[4] Because of these properties, the FraCaS data is not representative enough for system evaluation and comparison purposes.

Sec	Phenomenon	#Prob	#Mult-Pr	E	C	N	U
1	Generalized quantifiers	80	30	37	5	32	6
2	Plurals	33	9	20	5	8	0
3	(Nominal) anaphora	28	22	23	1	4	0
4	Ellipsis	55	30	41	3	11	0
5	Adjectives	23	8	9	6	7	1
6	Comparatives	31	15	19	3	9	0
7	Temporal reference	75	34	40	9	21	5
8	Verbs	8	0	6	0	2	0
9	Attitudes	13	4	8	1	4	0
	Total	346	152	203	33	98	12

Table 6.4: Stats of the FraCaS dataset (MacCartney and Manning, 2007): each section is characterized with the number of (multi-premised) problems and a distribution of gold answers, where E, C, N and U stand for ENTAILMENT, CONTRADICTION, NEUTRAL and UNDEFINED

For the above mentioned reason and the fact that few RTE systems are able to account for the phenomena in FraCaS or multi-premised problems, the data is rarely used for developing and assessing the semantic competence of RTE systems. Nevertheless, several RTE systems (MacCartney and Manning, 2008; Angeli and Manning, 2014; Lewis and Steedman, 2013; Tian et al., 2014; Mineshima et al., 2015) were trained and evaluated on (the parts of) the dataset. Usually the goal of these evaluations is to show expressive

[4] As RTE systems are deterministic classifiers they will always fail at least one such kind of problem, even if a system is able to solve both problems when a correct reading is indicated. One solution to accommodate ambiguous problems in RTE data set is to allow a list of answers as an answer. For instance, having [YES, DON'T KNOW] answer for FraCaS-87 would solve this issue.

Gold \ System	Positive / Proof		Negative No proof
	ENTAIL	CONTRAD	NEUTRAL
ENTAILMENT	**True** ENT	False CONT	False NEUT
CONTRADICT.	False ENT	**True** CONT	False NEUT
NEUTRAL	False ENT	False CONT	**True** NEUT

$$\texttt{Acc} = \frac{\texttt{TE} + \texttt{TC} + \texttt{TN}}{\texttt{All Problems}}$$

$$\texttt{Prec} = \frac{\texttt{TE} + \texttt{TC}}{\texttt{All Proofs}}$$

$$\texttt{Rec} = \frac{\texttt{TE} + \texttt{TC}}{\texttt{All Gold Posit}}$$

Table 6.5: A confusion matrix for three-way classification, related terminology and the definitions for the evaluation measures: accuracy, precision and recall

power and deep reasoning skills of specific theories or frameworks and the corresponding RTE systems. Given a FraCaS problem, whether a semantic framework and the system based on it are able to account for the phenomenon in the problem and correctly classify it consequently.

We have presented two very different RTE datasets, SICK and FraCaS. Both of them have relatively short sentences—the average number of words per sentence is 7.3 in FraCaS and 9.7 in SICK—and are three-way annotated. The datasets significantly differ in size and according to the phenomena they cover. The SICK problems represent relatively shallow reasoning over a pair of sentences while the FraCaS problems cover wide-range of semantic phenomena requiring deep reasoning over several premises. None of the datasets require encyclopedic knowledge (e.g., FC Barcelona is a football club). SICK contains no proper names and heavily employs lexical knowledge (e.g., a man is a person). In contrast to SICK, FraCaS contains proper names and requires no lexical knowledge as all required knowledge are provided by premises.

In the next subsections, we use the presented datasets for learning and evaluation purposes for the LangPro system. Employing both FraCaS and SICK datasets allows us to assess the performance of the prover on deep and relatively shallow reasoning respectively. While doing so, we employ accuracy, precision and recall as evaluation measures (see Table 6.5). Accuracy is a ratio of true guesses and total guesses. Precision is a fraction of positive guesses (i.e. proofs) that are correct, and recall is a fraction of positive problems that were classified correctly.

6.2 Learning

We first presented the natural tableau proof system with its inventory of rules and later, in this chapter, introduced the natural language prover which based on the system. Unlike this presentation order, in fact, we simultaneously developed the natural tableau proof system and LangPro. Here, we describe a data-driven process, which we call *training*, that basically represents how the LLF generator (Chapter 3) and the IR (Chapter 4) were enhanced in parallel with the prover. Training consists of two procedures. First is *adaptation* which concerns collecting the tableau rules, enriching the KB and designing the fixing rules. Second is *development*; it involves study of optimal parameters for the prover from the efficiency and the performance points of view.

6.2.1 Adaptation

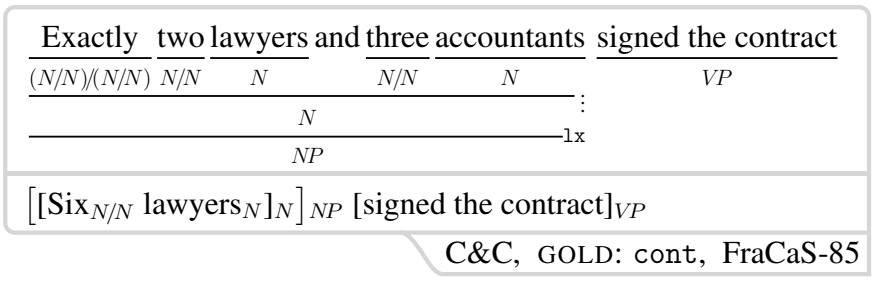
The adaptation procedure represents an incremental improvement of LangPro, which is driven by textual entailment problems found in the discussed RTE datasets. The procedure runs with intense assistance of an expert. In particular, given a set of the problems, each problem is automatically classified by the prover. If the prover misclassifies a problem, then the expert verifies the corresponding LLFs and tableau proofs, finds out where it went wrong and attempts to upgrade LangPro accordingly. A comprehensive description of the procedure is given in algorithm 4 (Appendix E). During the adaptation, we employ ccLangPro, i.e. LangPro using the C&C parser, and SICK-trail and FraCaS. From the latter data, we omit FrSec-4 as the prover has poor performance on sentences with ellipsis. While working with FrSec-3, 6 and 7, we only concentrate on improving LLFgen as the phenomena covered by the sections are not modeled in the natural tableau system.

In general, at least one of the components of LangPro—the C&C parser, LLFgen and NLogPro—is responsible for misclassification of a problem. The parser component is fixed: there is no upgrade of the prover if the parser *fails dreadfully*—it could not provide a derivation for one of the sentences in a problem or all returned derivations are not of the same syntactic category. For instance, (1) from FraCaS-36 could not be parsed while (2) from SICK-2911 was parsed but as of category VP, differing from the category of the corresponding hypothesis.

Every European is a person (1)

$\big[[\text{A woman}]_{NP}\big]_{S/VP}\ \big[\text{is putting}_{(VP/NP)/PP}\ [\text{on Makeup}]_{PP}\big]_{VP/NP}$ (2)

When one of the LLFs in a problem is not semantically adequate, sometimes it is possible to remedy it by introducing a new fixing rule in LLFgen. For example, in case of (FraCaS-85), we introduce the rules (11) and (12) in LLFgen. As a result, the obtained LLFs (3) and (4) are semantically adequate. This step corresponds to line 14 in the adaptation procedure.



It is not always reasonable to correct CCG derivations via new fixing rules. We only introduce the rules when they fix common errors in CCG derivations, e.g., the errors found in (FraCaS-85). Examples like the one in Figure 6.3, a CCG derivation for the premise of FraCaS-48, are not corrected as its remedy seems rather artificial and bound to a narrow case. The correction rules presented in §3.4 are collected in this way.

$$[\boldsymbol{c}^{\texttt{DT}|\texttt{CD}}_{\texttt{n,n}}\ \boldsymbol{N}_{\texttt{n}}]_{\texttt{np}} \rightsquigarrow \boldsymbol{c}_{\texttt{n,np}}\ \boldsymbol{N} \tag{11}$$

$$[\textbf{six}^{\texttt{CD}}_{\texttt{n,n}}\ \textbf{lawyer}_{\texttt{n}}]_{\texttt{np}} \rightsquigarrow \textbf{six}_{\texttt{n,np}}\ \textbf{lawyer}_{\texttt{n}}$$

$$[\boldsymbol{e}^{\texttt{RB}}_{(\texttt{n,n}),\texttt{n,n}}\ \boldsymbol{c}^{\texttt{CD}}_{\texttt{n,n}}\ \boldsymbol{N}_{\texttt{n}}]_{\texttt{np}} \rightsquigarrow \boldsymbol{e}_{(\texttt{n,np}),\texttt{n,np}}\ \boldsymbol{c}_{\texttt{n,np}}\ \boldsymbol{N}_{\texttt{n}} \tag{12}$$

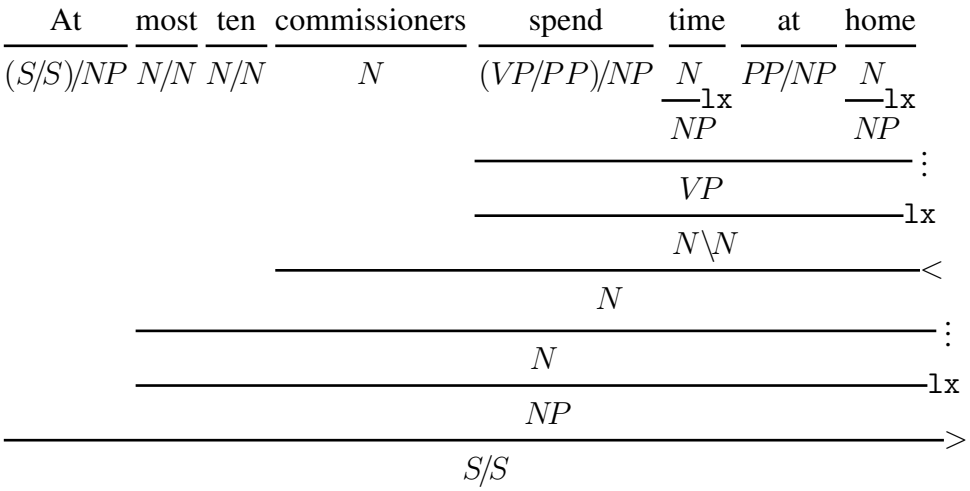


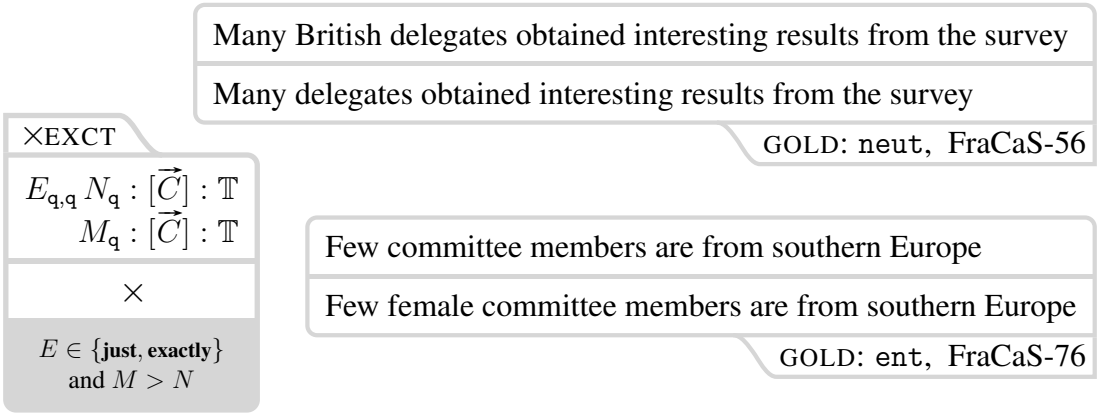
Figure 6.3: A CCG derivation from C&C that is useless to fix with LLFgen

$$[\textbf{exactly}^{\texttt{RB}}_{(\texttt{n},\texttt{n}),\texttt{n},\texttt{n}}\ \textbf{two}^{\texttt{CD}}_{\texttt{n},\texttt{n}}\ \textbf{lawyer}_{\texttt{n}}]_{\texttt{np}} \rightsquigarrow \textbf{exactly}_{(\texttt{n},\texttt{np}),\texttt{n},\texttt{np}}\ \textbf{two}_{\texttt{n},\texttt{np}}\ \textbf{lawyer}_{\texttt{n}}$$

$$\textbf{and}\ (\textbf{three}_{\texttt{q}}\ \textbf{accountant})(\textbf{exactly}_{\texttt{q},\texttt{q}}\ \textbf{two}_{\texttt{q}}\ \textbf{lawyer})\left(\lambda x.\ \textbf{the}_{\texttt{q}}\ \textbf{contract}\ (\lambda y.\ \textbf{sign}\ y\ x)\right) \quad (3)$$

$$\textbf{six}_{\texttt{q}}\ \textbf{lawyer}\ \left(\lambda x.\ \textbf{the}_{\texttt{q}}\ \textbf{contract}\ (\lambda y.\ \textbf{sign}\ y\ x)\right) \quad (4)$$

Semantically adequate LLFs are not always sufficient for a correct classification of a problem. As an example, consider FraCaS-85 and its LLFs (3) and (4) obtained from LLFgen after fixing the corresponding CCG derivations. It is possible to prove the contradiction relation in the problem if a new rule (×EXCT) is introduced in the IR. Therefore, we carry out line 22 and finally ccLangPro proves the problem as contradiction.



FraCaS-76 is an entailment problem that forces a new information to be added in the signature of NLogPro (see line 25). Namely, associating the downward monotonicity property with the first argument of **few** in the signature allows the prover to capture the desired entailment relation. Notice that the FraCaS problems favor to interpret "*few*" as an absolute amount rather than proportional. In the latter case, **few** would not have any monotonicity property in its first argument. On the other hand, there are FraCaS problems, e.g., FraCaS-56, that assume the proportional reading of "*many*", which is captured by modeling **many** as only being upward monotone in the second argument.

WordNet is an extensive lexical knowledge base, but obviously many relations are still absent there. For example, the lack of lexical knowledge is a reason for not accounting the entailments in SICK-4734, 5110. In such cases, we introduced required knowledge in the KB (see line 25), e.g., add **fit** $\sqsubseteq$ **appy** and **food** $\sqsubseteq$ **meal** to the KB. Notice that the prover extracts **meal** $\sqsubseteq$ **food** relation from WordNet as $meal_n^1$ is a hyponym of $food_n^1$ but not the required one. The relation **food** $\sqsubseteq$ **meal** is reasonable with "*chef*" but not in every context, e.g., "*I ate some food*" does not entail "*I ate some meal*". Since the theorem prover is purely compositional and does not take a context into account while computing semantics, adding relations like **food** $\sqsubseteq$ **meal** can be seen as a simple and yet reasonable way of accommodating entailments like SICK-5110. The

A man is *fitting* a silencer to a pistol
A man is *applying* a silencer to a gun
C&C, GOLD: ent, SICK-4734

A chef is preparing some *food*
A chef is preparing a *meal*
C&C, GOLD: ent, SICK-5110

During the adaptation procedure not all problems are solved by ccLangPro. We have already mentioned the problems where the parser fails dreadfully in analyzing sentences, e.g. (1) and (2), or where it is quite unreasonable to fix a CCG derivation; see an example in Figure 6.3. In order to prevent the prover from fitting the data or from unsoundness, we leave the problems like SICK-384 and 4505 unsolved. For instance, we could introduce **man** $\sqsubseteq$ **patient** relation to prove SICK-4505 (while **heal** $\sqsubseteq$ **help** is retrieved from WordNet), or add a corresponding rule or relation that accounts for the entailment from "*tall and green grass*" to "*field*" in SICK-384. The entailment problems where context plays a crucial role for lexical semantics, e.g. SICK-1584, are also troublesome for the prover.

A white and tan dog is running through the *tall and green grass*
A white and tan dog is running through a *field*
C&C, GOLD: ent, SICK-384

A *hole* is being *burrowed* by the badger
A badger is *shrewdly digging* the *earth*
C&C, GOLD: ent, SICK-1584

The doctors are *healing* a *man*
The doctor is *helping* the *patient*
C&C, GOLD: ent, SICK-4505

The adaptation is as a human-assisted learning procedure where the components of the prover are augmented: the IR with tableau rules, the KB – lexical semantic relations, the signature – algebraic properties of lexical entries and LLFgen – fixing rules. The problems in SICK-trial and FraCaS barely contribute to the signature and the KB respectively. This was expected as in general the SICK problems do not require reasoning over algebraic properties and FraCaS is self-contained in terms of knowledge. The problems with a neutral answer were hardly explored during the adaptation since the prover makes false proofs very rarely. This latter significantly decreases human work load.

6.2.2 Development

The theorem prover has certain parameters adjustment of which may lead to improvement of its performance. For example, what is the most optimal efficiency criterion for the prover? Which it better to consider single or multiple senses? Does the aligner component contributes to better performance? The development phase is used to answer such kind of questions. During the phase we employ ccLangPro to detect best parameter settings. For experiments we use separate development data: SICK-train (4500 problems). We consider those parameter settings as best for which the prover achieves the highest accuracy on the development data. While searching the best combination of parameters we use a greedy search—assuming that the locally best parameters guide to the globally best ones, or put differently, parameters are independent from each others.

We start the experiment with adjusting the number of WordNet senses per lexical entry. In § 5.1 we have discussed the single sense and multi sense approaches for the KB, where the former includes the most frequent sense approach $SS(1)$—giving each lexical entry the most frequent WordNet sense—and the latter associates every lexical entry with the multiple WordNet senses, also involving the option $SS(all)$ with all senses. To find the optimal solution for the sense parameter, we test ccLangPro with the following settings: $SS(1)$, $S(1\text{-}2)$, $SS(1\text{-}3)$, $SS(1\text{-}5)$ and $SS(all)$, where $SS(1\text{-}n)$ associates each word with its first n most frequent senses. Other parameters are set to its default value. The results show that more senses are under consideration higher is the accuracy on the data (see Table 6.6). For instance, $SS(all)$ improves upon $SS(1)$ and $SS(1\text{-}5)$ with 1.7% and 0.5% respectively. SICK-1251 and SICK-9350 are problems that require multiple senses. SICK-1251 is correctly classified with $SS(1\text{-}5)$ as $amalgamate_v^1$ and mix_v^5 are synonymous senses, but SICK-9350 requires more than the first five most frequent senses of "*rest*" because sit_n^1 is a hypernym of $rest_n^9$. The high number of considered senses also yields certain unsound proofs. For example, considering the second most frequent senses, the prover classifies SICK-1405 as entailment. On the other hand, access to $wall_n^7$—the sense of "*a masonry fence (as around an estate or garden)*"—enables the prover to prove SICK-1481 as entailment.

A woman is amalgamating eggs	Three women are resting in a village
A woman is mixing eggs	Three women are sitting in a village
GOLD: `ent`; SICK-1251	GOLD: `ent`; SICK-9350
A prawn is being cut by a woman	A deer is jumping over a wall
A woman is cutting shrimps	The deer is jumping over the fence
GOLD: `neut`; SICK-1405	GOLD: `neut`; SICK-1481

After finding out that the *all senses* approach maximizes the accuracy of ccLangPro, we test different strategies for rule application in the prover. For this, we check all 24 efficiency criteria for tableau rules.[5] In other words, we run ccLangPro with each permutation of the efficiency features, where $SS(all)$ is used for the KB and the rest of the

[5]Recover from § 5.2.1 that an efficiency criterion represents a permutation of the efficiency features of rules: non-branching, equivalence, non-consumer and non-producer. Based on the criterion and ($<_{pr}$), one can define the efficiency order ($<_{eff}$) over tableau rules (see § 5.3).

Acc%	Prec%	Rec%	Sense	Efficiency criterion	Aligner	RAL	Parser
75.09	98.5	43.6	1	[nonP, nonB, equi, nonC]	No	200	C&C
76.42	98.3	46.8	1-5	-	-	-	-
76.89	97.8	48.1	All	-	-	-	-
78.44	97.9	51.7	-	[equi, nonB, nonP, nonC]	-	-	-
79.33	97.9	53.8	-	-	Weak	-	-
81.5	97.7	59.0	-	-	Strong	-	-
81.53	97.7	59.1	-	-	Strong	400	-
81.38	98.0	58.5	-	-	Strong	400	EasyCCG
82.6	97.7	61.6	-	-	Strong	400	Both

Table 6.6: Results of the greedy search for the best settings of LangPro over SICK-train. The grey row represents the default settings of the prover used in the adaptation. Each omitted parameter borrows the value from the upper setting.

parameters are set to the default values. The results reveal that giving the highest priority to the non-producer feature leads to poor performance: 1.5% decrease in accuracy compared to the highest score. This is explained by the fact that in order to identify an inconsistency in a set of LLFs, one needs to introduce a sufficient number of entities in branches. Therefore, the introduction of fresh entities at early stage of a tableau is vital for the natural tableau theorem proving.

$$[\texttt{equi}, \texttt{nonBr}, \texttt{nonProd}, \texttt{nonCons}] \quad (5)$$
$$[\texttt{nonBr}, \texttt{equi}, \texttt{nonProd}, \texttt{nonCons}] \quad (6)$$
$$[\texttt{nonBr}, \texttt{nonProd}, \texttt{equi}, \texttt{nonCons}] \quad (7)$$
$$[\texttt{nonBr}, \texttt{nonProd}, \texttt{nonCons}, \texttt{equi}] \quad (8)$$

The prover achieved the highest accuracy with four criteria (5-8). We opt for (5)—favoring first the equivalence feature, then non-branching, non-producer and non-consumer in the end—since it produces slightly shorter tableau proofs than the others. For example, FraCaS-21 is the problem that is proved with these four criteria but not with the default one.

The residents of member states have the right to live in Europe
All residents of member states are individuals
Every individual who has the right to live in Europe can travel freely within Europe

The residents of member states can travel freely within Europe

GOLD: ent; FraCaS-21

The aligner component of the prover identifies and treats shared sub-terms of LLFs as constant terms (see §5.4). It excludes those sub-terms from aligning that are downward monotone or of type np which is not definite description, e.g., **no man** and **a man** are not treated as constants. Augmentation of the prover with the aligner component increases the accuracy score almost by 1% (see Table 6.6). The increase becomes larger (1.5%) if the rule application limit is decreased to 50. The aligner not only makes proofs shorter but also helps to find new proofs that the prover alone is not able to find even given an infinite

number of rule applications. The reason is that semantics of aligned LLFs are much simpler than those of non-aligned ones. For example, with the help of the aligner, the prover solves SICK-1022 and SICK-727, where the underlined constituents correspond to the aligned sub-terms.[6] Notice that in SICK-train there is no problem that is proved using non-aligned LLFs but not proved with aligned ones. This fact shows that in the SICK problems the semantics of shared constituents are irrelevant.

A woman is wearing sunglasses of large size and is holding newspapers in both hands
There is no woman wearing sunglasses of large size and holding newspapers in both hands
GOLD: cont; SICK-1022

The man in a grey t-shirt is sitting on a rock in front of the waterfall
There is no man in a grey t-shirt sitting on a rock in front of the waterfall
GOLD: cont; SICK-727

The aligner avoids to align upward monotone GQs if they do not represent definite descriptions or coordinations of them (e.g., "*the boy and the girl*"). Hence, the GQs for definite NPs or bare plurals are not aligned. On the other hand, there are many problems in SICK assuming that the indefinite, plural or collective NPs shared by a premise and the conclusion co-refer; see SICK-90 and 423 for the demonstration of it. From the orthodox semantics perspective, endorsed by LangPro, these two problems do not necessarily encode contradictions. Imagine a situation where there are two men and two swimming pools, one empty and another full. If each man jumps in a different pool, then the situation makes both sentences true. The similar counterexample exists for the second problem too.

A man is jumping into *an empty pool*
A man is jumping into *a full pool*
GOLD: cont; SICK-90

Two men are not holding *fishing poles*
Two men are holding *fishing poles*
GOLD: cont; SICK-423

One option to prove such problems is to force entities to be equal if they satisfy the same common noun, in other words, to treat all NPs as definite descriptions. Unfortunately, this approach comes with many false proofs too. One might think of capturing contradictions like SICK-423 by giving a wide scope to the negation in LLFs. We do not opt for this approach as it requires complication of LLFgen and is tailored for specific problems. To tackle this issue partially, we adopt a simple solution. We make the aligner stronger by allowing it to align all NPs that are not downward monotone. Hence, SICK-423 is proved in a straightforward way. On the other hand, the problems

[6] Without the aligner, a tableau for SICK-1022 does not terminate because in the LLF of the premise **both hand** takes scope over **s newspaper** and **both** is treated as a universal quantifier. Therefore, without the knowledge **newspaper**|**hand** there are an infinite number of entities introduced in the tableau: for some *hand* introduce some *newspaper* and if the latter is a hand, then introduce some another *newspaper*, etc. Due to the shortcoming of syntactic types (see § 3.5), in the LLFs of SICK-727, **a grey t-shirt** takes scope over **the man** and **no man**. As a result, the existence of some *grey t-shirt* worn by some man with the property P does not contradict the existence of another *grey t-shirt* that is worn by no man with the property P.

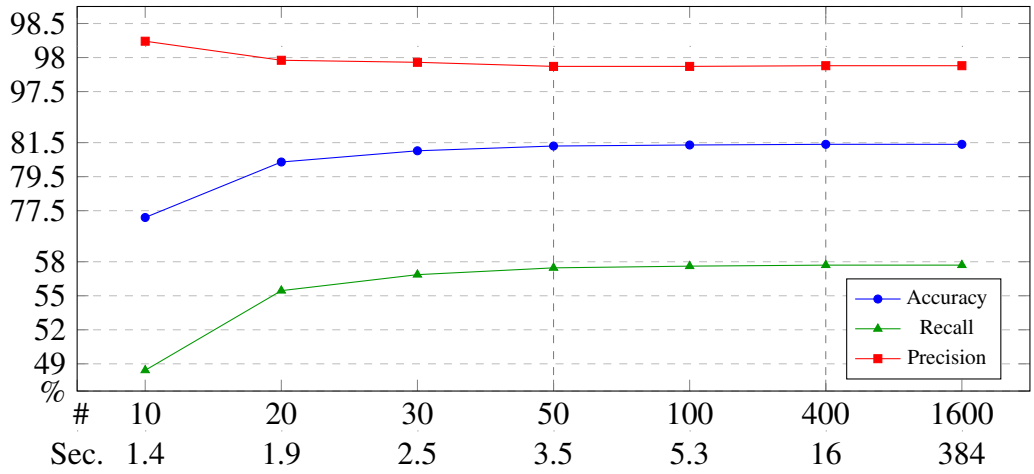


Figure 6.4: Performance of LangPro (with the aligner and C&C) on SICK-train for different rule application limits. Average runtime in seconds per 100 problems is based on an 8×2.4GHz CPU machine. NB different scales for each measure on the Y axis.

like SICK-90 stay unsolved, which we plan to address in future work. After making the aligner stronger, the accuracy of the prover with the aligner increases significantly (see Table 6.6). This shows that the annotation of the dataset mainly endorses co-reference of the same NPs.

Up to now, the number of rule applications in each tableau was limited to 200. To find out how the performance of the prover depends on the RAL, we run the prover with different values of RAL. The results are given in Figure 6.4 with average runtimes per 100 problems. The experiment shows that the RAL of 400 is an effective upper-bound: if a proof is not found in 400 rule applications, then it will not be found in 1600 rule applications either.[7] On the other hand, the RAL of 50 is efficient: the provers with the RAL of 50 and of 1600 differ only in five problems (SICK-9554 is one of those). These five problems were not proved with 50 rule applications because the shared constituents in the problems were parsed differently by the C&C parser, e.g., see the agent of the verb in SICK-9554. Such differences apparently require many rule applications to be discovered and analyzed similarly.

A group of [children in a church basement]$_{NP}$ is playing maracas and tambourines
Tambourines are being played by [a group of children]$_{NP}$
GOLD: `ent`; SICK-9554

In addition, for the SICK dataset we found out that intersective treatment of nominal modifiers, e.g., adjectives, increases the performance. For this reason the results in Table 6.6 already use intersective treatment for the modifiers by default. If only those intersective adjectives that are hard-coded in the signature are treated intersectively, then the accuracy (81.53%) of the best configuration drops by 0.2%. We also tested the two

[7] In particular, the prover with 400 and 1600 RALs achieves the same results. The same is true for the RALs of 100 and 200.

approaches for definite NPs presented in §4.3.2: one treating "*the*" as the indefinite determiner "*a*" and another treating definite NPs as referring expressions. The latter approach works better for FraCaS while for SICK-train the former approach shows slight gain of 0.09% in accuracy. Since the improvement seems insignificant, for both datasets we treat definite NPs as referring expressions.

Borrowing the best parameter setting from ccLangPro, we try it for easyLangPro—the language prover based on the EasyCCG parser (Lewis and Steedman, 2014a). easyLangPro scores accuracy of 81.16%, comparable to ccLangPro (81.4%). This show that the prover, and particularly LLFgen and the IR, are not overfitting the C&C derivations and can use any CCG derivation successfully that is compatible with the CCGbank format (Hockenmaier and Steedman, 2007).

Distinct judgments of ccLangpro and easyLangPro are conditioned by the differences between the CCG derivations of C&C and EasyCCG (discussed in more details in §6.3). In order to abstract LangPro from parsing mistakes to some extent, we combine the answers of both provers based on different parsers. The combination is simple, if one of the parser-specific provers classifies a problem positively, i.e. entailment or contradiction, then let this positive judgment be the final guess; otherwise the problem is classified as neutral. An exception applies to the case when the provers return different positive answers. In this situation the combined answer is neutral. The combined prover improves upon both ccLangPro and easyLangPro with more than 1% gain in accuracy (see Table 6.6). The increase is achieved by the boost in the recall while the precision stays almost the same. There was no problem for which the parser-specific prover returned different positive answers—for each of the problems at least on of the provers judged it as neutral. In the context of the parser-specific provers, we refer the combined prover as coLanPro; but when talking about the performance of LangPro in general, we mean coLangPro as the latter can be regarded as representative of the language prover that least depends on a specific CCG parser.

We have described the development phase of LangPro on SICK-train. The phase includes the adjustment of the following parameters: the number of WordNet senses per lexical entry, an efficiency criterion, augmentation with the aligner and its strength, the rule application limit and the parser component. Following the greedy search, the highest accuracy was achieved by coLangPro, a combination of ccLangPro and easyLangpro, when all the WordNet senses are under consideration, the strong aligner is incorporated, the most efficient criteria is chosen and the RAL is set to 400.

6.3 Analysis of the results

In the previous section, we gave the results the provers obtain on the development data. But what is the actual story behind the results? Why there are proves obtained for neutral problems? What stops the provers from better performance? To answer the similar questions, we will explore entailment judgments from the versions of LangPro (see Table 6.7 and Table 6.8) and see what are the reasons behind both incorrect and correct guesses. Below we will do so by describing in details how the prover processes particular entailment problems drawn from SICK-trial, SICK-train and FraCaS. For each presented problem, along with its ID and gold answer, we will give two judgments each corresponding to the

Gold \ Prover	ccLangPro				easyLangPro				**coLangPro**		
SICK-train	E	C	N	F	E	C	N	F	E	C	N
Entailment	701	0	589	9	692	0	607	0	725	0	574
Contradiction	0	459	205	1	0	457	208	0	0	485	180
Neutral	16	11	2483	26	11	12	2513	0	16	13	2507

Table 6.7: Confusion matrices of the versions of LangPro on SICK-train. The F columns count the problems where the parsers failed dreadfully.

Gold \ Prover	ccLangPro				easyLangPro				**coLangPro**		
FraCaS	E	C	N	F	E	C	N	F	E	C	N
Entailment	51	0	23	4	52	0	22	0	59	0	15
Contradiction	1	14	2	0	1	12	4	0	1	14	2
Neutral	1	0	50	6	2	0	49	0	2	0	49

Table 6.8: Confusion matrices of the versions of LangPro on the applicable FraCaS sections 1, 2, 5 and 9. The F columns count the problems where the parsers failed dreadfully.

answer of a parser-specific prover. The following terminology will be used. We refer the LLFs obtained from the C&C or the EasyCCG derivations as ccLLFs and easyLLF respectively. Their aligned versions are denoted by $\overline{\text{cc}}$LLFs and $\overline{\text{easy}}$LLFs. When we want to emphasize a decision made by one of the parser-specific provers that was influenced by the aligner, we talk about $\overline{\text{cc}}$LangPro and $\overline{\text{easy}}$LangPro. The selected set of examples intends to highlight the issues of natural language theorem proving and give the clues for further improvements.

6.3.1 True entailments and contradictions

First, we are going to discuss the positive problems (i.e. having the entailment or contradiction gold label) that were correctly proved either by ccLangPro or easyLangPro. Eventually such problems are called true positives as coLangPro correctly guesses them.

CC/Easy: Men are sawing$_{VP/NP}$ [logs$_N$]$_{NP}$

CC: There are no $\left[\text{men}_N\ [\text{sawing}_{VP}]_{N/N}\right]_N$ **Easy**: There are no [men$_{N/N}$ sawing$_N$]$_N$

cc/easyLP: C/N G: C SICK-1417

SICK-1417: The problem involves verb subcategorization, the expletive "*there*" and the negative universal quantifier "*no*". Both parsers correctly analyze the premise while in case of the conclusion only EasyCCG makes a mistake: "*men*" is modifying "*men*" rather than vice versa. This mistake of EasyCCG is crucial for easyLangPro as it prevents the prover from finding the proof for contradiction. Fortunately, correct C&C derivations result in semantically adequate ccLLFs which are proved by LangPro as inconsistent in 10 rule applications.
Conclusion: The proof is not found with EasyCCG due to a wrong derivation but the C&C derivations salvage the situation.

The girl [in blue]$_1$ is chasing the base runner $\big[$with a number [on the jersey]$_3\big]_2$
The girl [in blue]$_1$ is chasing the player $\big[$with a number [on the jersey]$_3\big]_2$
cc/easyLP: E/E G: E SICK-8147

SICK-8147: The key in the problem is to regard "*base runner*" as "*player*", but it contains sentences challenging for the parsers. In particular, both sentences contain three PPs separately, marked with brackets and indexed. Apart from the different analyses for each NP,[8] the derivations from C&C and EasyCCG also differ in PP attachments. For both sentences, C&C treats PP_3 as an argument of "*number*" while EasyCCG analyzes it as a modifier of "*a number*". In both sentences, EasyCCG wrongly but in a consistent way treats PP_2 as a VP modifier. PP_2 gets mixed analyses from C&C: correctly identified it as a modifier of "*player*" in the conclusion but analyzed wrongly in the premise, similarly to EasyCCG. Due to these differences, the corresponding ccLLFs and easyLLFs, including their aligned versions, also differ from each other. More specifically, the terms for PP_2 are aligned in the $\overline{\text{easy}}$LLFs; but in the $\overline{\text{cc}}$LLFs, the shorter sub-terms of "*a number on the jersey*" are aligned because C&C analyzes PP_2 in the inconsistent ways and assigns different categories to "*with*" in the sentences.

For both versions of aligned LLFs, the prover finds proofs for the entailment relation. Due to the poor alignment for the $\overline{\text{cc}}$LLFs, the tableau was closed in 20 rule applications while for the $\overline{\text{easy}}$LLFs, with a better alignment, in 8 rule applications. Despite the wrong attachments of PP_2 in the $\overline{\text{easy}}$LLFs, proof search is more efficient because the attachments were *consistent* which contributed to the better alignment. In case of the $\overline{\text{cc}}$LLFs, the attachments of PP_2 were inconsistent (though one of them was correct) which finally costs much in terms of a lengthy proof. Moreover, finding a proof for the $\overline{\text{cc}}$LLFs would be impossible if we did not introduce ($\times\text{PP@V}_{\mathbb{T}}$) in §4.2.2, which helps to abstract the $\overline{\text{cc}}$LLFs from the inconsistency in PP attachments. As a result, the rule identifies the nodes in ($\times\text{PP@V}'_{\mathbb{T}}$) as inconsistent: the terms written in CamelCase are the corresponding aligned sub-terms while the individual constant c stands for "*the base runner*".

$\times\text{PP@V}_{\mathbb{T}}$

$$\frac{[p^{\texttt{IN}}\, t_e] : V_{(-,\texttt{s})} : [\vec{A}] : \mathbb{T} \qquad p^{\texttt{IN}}_{\texttt{np,pp}} : [t, c] : \mathbb{F}}{\times} \quad c \in \vec{A}$$

$$\frac{[\textbf{with NumOnJers}] : \textbf{chase} : [c, \textbf{GirlInBlue}] : \mathbb{T} \qquad \textbf{with} : [\textbf{NumOnJers}, c] : \mathbb{F}}{\times}\,(\times\text{PP@V}'_{\mathbb{T}})$$

Knowing **base runner** $\sqsubseteq$ **player** is a main part of solving SICK-8147. This piece of knowledge is necessary for the tableau to close. Fortunately, the synset of $baserunner^1_n$ is an indirect hyponym of the synset of $player^1_n$ in WordNet. Therefore, the necessary semantic relation in found in the KB and the tableau closes.[9]

[8] EasyCCG usually analyzes NPs with post-modifiers in the NP-S style, i.e. a determiner is grouped with its head earlier than post-modifiers: $\big[$[the girl]$_{NP}$ [in blue]$_{NP\backslash NP}\big]_{NP}$. On the other hand, C&C trained on rebanked CCGbank (Honnibal et al., 2010) prefers the Nom-S analysis: $\big[$the$_{NP/N}$ [girl$_N$ [in blue]$_{N\backslash N}$]$_N\big]_{NP}$.

[9] In case of the multi-sense approach for the KB, there is alternative option for the prover to access

Conclusion: The consistent (possibly wrong) analyses of PP attachments leads to the better aligned LLFs, which itself contributes to shorter proofs. A wrongly attached structurally ambiguous PP can be identified and correctly interpreted with the help of the special tableau rules (presented in § 4.2.2). The (single) most frequent sense approach is sufficient for the current entailment problem.

Every European has the right $\big[$[to live in Europe]$_{VP_{to}}\big]_{N\backslash N}$ Every European is a person Every person who has the right [to live in Europe] can travel freely within Europe
Every European can travel freely within Europe
cc/easyLP: N/E G: E FraCaS-18

FraCaS-18: The textual entailment problem contains multiple premises but this is not a problem for LangPro. A challenge in this example is to obtain decent derivations and to convert them in LLFs. The C&C-based prover fails in the beginning when the parser fails to return the derivations for the first two sentences, which contain relevant information for the entailment. Fortunately, EasyCCG gets all the sentences parsed. The produced easyLLFs are not proper λ-terms since LLFgen, at this moment, does not have a remedy for the type changing combinatory rule which converts the syntactic category VP_{to} into N/N. Despite this shortcoming, LangPro is still able to operate on the easyLLFs and find a proof for entailment in 43 rule applications (actually, only a few of those applications contribute to the proof). If decomposition of the LLF for "*the right ... Europe*" was necessary for the proof, due to the unexplained type-change, the prover would fail to do so and would not find the proof. Notice that in this example, the aligner is useless as there is no constituent phrase shared by all the sentences.
Conclusion: C&C failed to parse several sentences, however EasyCCG saved the situation. The obtained easyLLFs are not well-formed terms; nevertheless the prover is able to process them as long as the decomposition of the ill-formed sub-terms is not necessary for the proof.

[A woman]$_{NP}$ is not talking on [a telephone]$_{NP}$
[A woman]$_{NP}$ is talking on [a telephone]$_{NP}$
cc/easyLP: C/C G: C SICK-1207

SICK-1207: The problem represents a dubious case. One might identify the problem as neutral or contradiction depending whether the indefinite NPs "*a woman*" and "*a telephone*" are referencing to the same referents respectively. The annotation in SICK show a strong tendency towards co-reference of the same indefinite NPs; the similar problems, SICK-363 and 1989, with the similar judgments are given in Table 6.9. Co-reference of the identical indefinites is simply achieved with the help of the strong aligner. The aligned

base runner $\sqsubseteq$ **player**: first capture **base runner** $\sqsubseteq$ **runner** using a rule for subsective adjectives and then combine it with **runner** $\sqsubseteq$ **player**, which is retrieved from WordNet as $baserunner^1_n$ and $runner^4_n$ are in the same synset. Notice that the multi-sense approach also allows an unwanted entailment from "*A runner won*" to "*A player won*" due to **runner** $\sqsubseteq$ **player**.

LLFs for the problem are given in (9) and (10). Using the aligned LLFs, the prover is able to find a proof for contradiction in 3 rule applications.

$$\textbf{not}_{\mathtt{vp},\mathtt{vp}}\ \textbf{beTalkOnATelephone}_{\mathtt{vp}}\ \textbf{aWoman}_{\mathtt{np}} \tag{9}$$

$$\textbf{beTalkOnATelephone}_{\mathtt{vp}}\ \textbf{aWoman}_{\mathtt{np}} \tag{10}$$

Conclusion: the simple solution with the alignment technique accounts enough well for the co-reference of the same indefinites. It also makes proofs extremely short.[10]

We have presented the problems that were correctly classified by the prover in spite of the various shortcomings. Usually this kind of problems is rare but interesting. For instance, from the four presented examples, the judgments of ccLangPro and easyLangPro diverged in half of the cases, but based on the development set—SICK-train (4500 problems)—the judgments diverge only for 2.9% of the problems.

6.3.2 False entailments and contradictions

Given an almost perfect precision of the prover, the false positive problems represents a special case of interest.

A man with no hat [is sitting on the ground]$_1$
A man with a backwards hat [is sitting on the ground]$_1$
cc/easyLP: C/C G: N SICK-8461

SICK-8461: The problem is similar to SICK-1207 in terms of possibly co-referring definite NPs, but it is labeled as neutral by majority of annotators. The ccLLFs and easyLLFs for both sentences are quite similar; the ccLLF and the easyLLF for the premise are given in (11) and (12), respectively. The only difference is in the analyses of the verb phrases: whether **be** takes only **sit** or the whole verb phrase as an argument. This difference has no influence on proof search since the auxiliaries are currently treated as identity functions.

$$\textbf{no hat}\ \lambda y\big(\textbf{a}\ \big(\textbf{with}\ y\ \textbf{man}\big)\ \lambda x\big(\textbf{the ground}\ \lambda z(\textbf{on}\ z\ (\textbf{be sit})\ x)\big)\big) \tag{11}$$

$$\textbf{no hat}\ \lambda y\big(\textbf{a}\ \big(\textbf{with}\ y\ \textbf{man}\big)\ \big(\textbf{be}\ \lambda x(\textbf{the ground}\ \lambda z(\textbf{on}\ z\ \textbf{sit}\ x))\big)\big) \tag{12}$$

$$\mathtt{a}_{(et)(et)t}\ \lambda y\big(\mathtt{no}_{(et)(et)t}\ \mathtt{hat}_{et}\ \lambda x(\mathtt{with}_{e(et)et}\ x_e\ \mathtt{man}_{et}\ y_e)\big)\ \mathtt{SitOnGround}_{et} \tag{13}$$

In contrast to the surface level, **no hat** takes the widest scope in the LLFs. The reason is the usage of terms with syntactic types. While using the terms of semantic type, it is possible that **no hat** takes a narrow scope, see (13). But in case of the syntactic types, **no hat** cannot be type-raised in the PP because a determiner of type $(\mathtt{n},(\mathtt{np},\mathtt{s}),\mathtt{s})$ cannot take a

[10] If in the premise "*a woman*" is replaced by "*a person*", the alignment approach cannot contribute to the proof. More general solution to the co-reference of indefinites is achieved when the negation takes a wide scope. Implementation of the latter approach becomes complex when considering sentences with several clauses. It also does not go hand in hand with reasoning on surface forms. The co-reference of indefinites without the aligner is left for future work.

term of type $(\mathtt{np}, t)$ or (e, t) for its second argument; remember that $(\mathtt{np}, t)$ and (e, t) are not subtypes of $(\mathtt{np}, \mathtt{s})$.[11]

It is obvious that if the sentences were understood with "*no hat*" and "*a backwards hat*" having the widest scope, then they would be inconsistent. This is why the prover classifies the problem as contradiction. The proofs for both versions of LLFs were found in 5 rule applications. The alignment of VP_1 does not affects the proof search as the relevant terms "*no hat*" and "*a backwards hat*" are analyzed and contrasted to each other before VP_1 is processed.

Conclusion: The natural order for the quantifier scopes is not obtained due to less-flexible syntactic types, which in the end leads to the prediction different from the gold label. This mistake of the prover seems minor taking into account that the similar problems, e.g., SICK-8562 in Table 6.9, receive mixed judgments (neutral or contradiction) by the SICK annotators.

There is $\big[$[no man] and [child kayaking through gentle waters]$\big]$
A man and a young boy are riding in a yellow kayak
cc/easyLP: C/C G: N SICK-7402

SICK-7402: The problem is neutral as the sentences are informative with respect to each other, but both parser-specific provers identify it as contradiction. The reason is wrong CCG derivations where "*no*" takes scope only over "*man*". In this way, the premise entails that there is no man while the conclusion asserts the contrary—a man is riding in a kayak. The provers detect this inconsistency and classify the problem as contradiction.

Conclusion: The mistakes by the C&C and EasyCCG parsers misled the prover. In general, the parsers do many mistakes but it is very rare that they lead the prover to a false proof, like in the recent example.

$\big[\text{At}_{(S/S)/NP}\ [\text{most}_N]_{NP}\big]_{S/S}\ \big[$ten female commissioners spend time at home$\big]_S$
$\big[\text{At}_{(S/S)/NP}\ [\text{most}_N]_{NP}\big]_{S/S}\ \big[$ten commissioners spend time at home$\big]_S$
cc/easyLP: N/E G: N FraCaS-64

FraCaS-64: Since "*at most* n" is downward monotone for any number n, the problem does not represent entailment. The example contains sentences that happen to be difficult for the parsers. C&C dreadfully fails to analyze both sentences, assigns the S/S category to both derivations while EasyCCG is able to return the derivations of category S. In spite of the wrong CCG derivations, it is possible to block the entailment for the easyLLFs given that the sentential modifier "*at most*" is not monotone. As there is no information about monotonicity properties of compound terms in the signature, the term for "*at most*" by default is regarded as upward monotone. As easyLangPro is able to prove the entailment relation between the arguments of "*at most*", it classifies the problem as entailment.

Conclusion: EasyCCG returned wrong derivations but still better ones than those of C&C. It is simply unfortunate that the wrong derivations and the absence of monotonicity properties of compound terms in the signature result in the proof for entailment.

[11] This issue can be solved by introducing a semantic counterpart of the determiner that is of type $(et)(et)t$, but this itself will further require introduction of semantic counterparts of other terms. The latter approach almost doubles the number of lexical terms in a tableau which consequently complicates a proof search.

ID	G/LP	Premise	Conclusion
1405	N/E	A **prawn** is being cut by a woman	A woman is cutting **shrimps**
1481	N/E	A deer is jumping over a **wall**	The deer is jumping over the **fence**
1777	N/E	A **boy** is happily playing the piano	A piano is being played by a **man**
4443	N/E	A man is singing to a **girl**	A man is singing to a **woman**
2870	N/C	Two people are riding a **motorcycle**	Nobody is riding a **bike**
2868	E/N	Two people are *stopping* on a **motorcycle**	Two people are *riding* a **bike**
6258	E/N	A *policeman* is sitting on a **motorcycle**	The cop is sitting on a *police* **bike**
344	N/C	P: An Asian woman in a crowd is not carrying a black bag C: An Asian woman in a crowd is carrying a black bag	
545	N/C	P: A woman is standing and is not looking at the waterfall C: A woman is sitting and looking at the waterfall	
8913	N/C	A couple is not looking at a map	A couple is looking at a map
363	C/C	A soccer ball is not rolling into a goal net	A soccer ball is rolling into a goal net
1989	C/C	A girl is playing the guitar	A girl is not playing the guitar
8562	C/N	P: A man *in a hat* is standing outside of a green jeep C: A man *with no hat* is standing outside of a green jeep	

Table 6.9: The false positive examples and the problems with noisy gold (G) labels. The problems are drawn from SICK-train. The words that were related to each other by LangPro (LP) are in bold while the unrelated ones in italic.

A person is folding a sheet

A person is folding a piece [of paper]$_1$

cc/easyLP: E/E G: N SICK-5264

SICK-5264: For the sentences of the problem, there exist readings that validate the entailment relation, but the majority of annotators does not consider those readings and classify the problem as neutral. The decision of the prover, whether based on C&C or EasyCCG, does not coincide with the gold label. Different natures of attachment for PP$_1$—as a noun argument by C&C and as an NP modifier by EasyCCG—do not affect the final judgments because all the argument PPs are also treated as modifier PPs with the help of the tableau rules (see §4.2.2).

The reason for the proof for entailment is the relation **sheet** $\sqsubseteq$ **paper** in the KB retrieved from WordNet (the synset of $sheet_n^2$ is a hyponym of the synset of $paper_n^1$) and the tableau rule that identifies "*a piece of paper*" as "*paper*". The entailment is proved in 18 rule appreciations.

Conclusion: The problem is ambiguous but has the neutral gold label. The multi-sense approach to the KB allows the readings of the sentences that lead the prover to the proof for entailment.

The other false positives that are proved in the same vein as SICK-5264 are give in the upper part of Table 6.9. The problems SICK-1405, 1481 and 1777 are classified as entailment due to the lexical relations **prawn** $\sqsubseteq$ **shripm**, **wall** $\sqsubseteq$ **fence** and **boy** $\sqsubseteq$ **man**, respectively, licensed by the multi-sense approach. For proving SICK-4443 and 2870, the single sense approach with the most frequent sense is sufficient. Notice noise in gold labels with respect to **motorcycle** $\sqsubseteq$ **bike** relation. While SICK-2870 rejects it, SICK-2868

and 6258 presuppose the relation. Unfortunately, LangPro was not able to capture the latter two entailments as it failed to relate other lexical entries.

On the SICK dataset, the prover rarely finds false proofs and when it does, the multi-sense approach or the noisy labels are the reason in around 70% of the cases. Apart from FraCaS-64, other false proofs of the FraCaS problems involve ambiguous problems—two identical problems having different gold answers due to the ambiguity of involved sentences. For example, one of the two instances of such ambiguous problems are FraCaS-88 and 109. Since LangPro is a deterministic system, it always fails one of the instances of ambiguous problems. It is able to correctly classify the counterparts of FraCaS-88 and 109 as entailments. FraCaS-109 is the only problem when the prover confuses entailment and contradiction.

Every representative and client was at the meeting
Every representative was at the meeting
cc/easyLP: E/E G: N FraCaS-88

Just one accountant attended the meeting
Some accountants attended the meeting
cc/easyLP: E/E G: C FraCaS-109

6.3.3 False neutrals

There can be several reasons for a false neutral: starting from the mistakes by the CCG parsers finishing with a lack of knowledge. The prover shows a large number of false neutrals on SICK. In order to find out the reason behind it, we randomly drew 200 problems from SICK-train and classified the false neutrals found there (see some of the examples in Table 6.10). Around a half of the false neutrals were due to knowledge sparsity. For example, in order to prove the entailment in SICK-4974, one needs to know **hedgehog** $\sqsubseteq$ **small animal**, which is not available from WordNet. A lack of tableau rules was a reason for a quarter of the problems. This kind of problems also include the cases where an absent paraphrase can be captured with a schema, e.g., X *made of* $Y \rightarrow YX$, like in SICK-4553. The problems concerning cardinality also need assist through tableau rules. The rest of the false neutrals were evenly triggered by noisy gold labels and the mistakes coming from the parsers. For instance, in SICK-4720, "*monkey*" and "*chimp*" are disjoint concepts while from a layman's perspective "*monkey*" is often regarded as a superset of "*chimp*". In SICK-6447, both parsers choose wrong constituents which in the end make impossible even for the aligner to align the sub-terms for identical phrases.

There are several reasons behind the FraCaS problems that were not proved by the prover. Several problems for adjectives were not proved as they contained comparative constructions, not covered by the prover and the natural tableau. Some problems assume the universal reading of plurals. A couple of problems involving *at most* were not solved as the parsers often analyze the phrase in a wrong way.

In order to highlight the cases where the prover fails or succeeds, we have presented concrete entailment problems and explained the judgments of both parser-specific provers

ID	G	Fail	Premise	Conclusion
4720	E	G	A *monkey* is practicing martial arts	A *chimp* is practicing martial arts
4275	E	R	*A man and a woman* are shaking hands	*Two persons* are shaking hands
4553	E	R	P: A man is emptying a *container made of plastic* C: A man is emptying a *plastic container*	
2763	C	K	A man and woman *are talking*	A man and a woman *are silent*
4974	E	K	Someone is holding a *hedgehog*	Someone is holding a *small animal*
6447	C	P	P: [A small boy [in a yellow shirt]] is laughing on the beach C: There is no small boy [in a yellow shirt [laughing on the beach]]	

Table 6.10: Examples of false neutrals from SICK. The factors for the failure (Fail) are noisy gold labels (G), the mistakes by the parsers (P), a lack of rule (R) and a lack of knowledge (K). Each problem is marked with the reason of failure.

for it. The discussed examples involve true and false proofs and false neutrals drawn from SICK-train and FraCaS. False positives are due to errors in the CCG derivations, ambiguous entailment problems or the multi-sense approach. The reasons behind false neutrals are

6.4 Evaluation & comparison

After the learning phase, we are ready to evaluate the prover against the datasets and compare the results to related systems. Evaluation is carried out on the relevant sections of FraCaS and SICK-test, the latter being unseen during the adaptation and the development. For the evaluation LangPro uses the best configuration that was obtained after the learning. In particular, it represents a combination of ccLangPro and easyLangPro, where both provers employ WordNet, all-sense approach, the efficient criterion (5), the strong aligner and the rule application number of 400. Apart from quantitative comparison, we also make qualitative comparison of our prover with related systems.

6.4.1 Based on FraCaS

We have adapted the prover to the FraCaS sections for generalized quantifiers (GQs), plurals, adjectives and attitudes.[12] The evaluation measure of coLangPro for each applicable section are presented in Table 6.11 while the confusion matrix is in Table 6.8. There are only three false proofs among the FraCaS problems. They are due to wrong parsing and ambiguous sentences (see §6.3). According to the results, reasoning over monotonic features of GQs comes easily to the prover. Only four problems from 74 were classified wrongly: CCG derivations are responsible for three of them while one problem was not proved as it assumed the universal reading of a bare plural.

The section for plurals represents a tough portion for the prover as it contains ambiguous conjoined noun phrases, e.g., "*Exactly two lawyers and three accountants*", and

[12]The section for ellipsis was dropped out due to poor performance of the CCG parsers. The rest of the sections for anaphora, comparatives, temporal reference and verbs were omitted as these phenomena are not modeled in the natural tableau system yet. For instance, in case of temporal reference and verbs, one needs to model time and aspect in the natural tableau system.

Sections	#Prob	coLangPro			−HOGQ
		Prec%	Rec%	Acc%	Acc%
1 GQs	74	98	93	95	89
2 Plurals	33	89	86	73	70
5 Adjectives	22	100	67	77	77
9 Attitudes	13	100	89	92	92
1,2,5,9	142	96.1	81.3	86.6	82.4

Table 6.11: Evaluation of coLangPro and coLangpro$^{-\text{HOGQ}}$ for each applicable section

different readings of bare plurals, e.g., universal, quasi-universal (i.e. almost every) and strictly more than one. There are also problems with the words, e.g. "*another*" and "*likely*", semantics of which are not yet handled properly by the prover. The prover accounts for subsective, intersective and privative adjectives in a simple and adequate manner. Wrongly classified problems about adjectives are due to comparative constructions found there. Only one problem in the section of attitudes was not proved. The reason is mistakes comping from the parsers. [13]

An advantage of a rule based system with high accuracy is that it can be used to evaluate data. In particular, with the help of the prover, we check the FraCaS section on how representative they are for higher-order GQs (HOGQs). To do so, we replaced all occurrences of **most**, **several**, **many**, **s** and **the** with the indefinite **a** in LLFs. The GQs with downward monotone properties, e.g., **few**, **every** and **no**, were left intact. In this way we obtain the prover LangPro$^{-\text{HOGQ}}$ which is incapable to properly handle HOGQs. Despite this shortcoming, LangPro$^{-\text{HOGQ}}$ achieves overall accuracy of 82.4% over the applicable sections (see Table 6.11). Compared to LangPro, only six problems were misclassified: five and one from the sections 1 and 2 respectively. If we do not substitute **the**, which allows co-reference of certain NPs, from these six problems four are solved—increasing the accuracy to 85.2%. In this way, only FraCaS-56 and 58 are sensitive to this change, which are falsely proved by LangPro$^{-\text{HOGQ}}$. This shows that FraCaS, despite its dedicated section to GQs, is not representative enough for HOGQs, like "*many*" and "*most*". A system which does not distinguish indefinite NPs from these HOGQs can also achieve high results on the dataset.

Many British delegates obtained interesting results from the survey

Many delegates obtained interesting results from the survey

GOLD: `neut`; FraCaS-56

[13]The current version of coLangPro differs from coLangPro*, the version of the prover presented in Abzianidze (2016), only in two aspects: the usage of the efficiency criterion and treatment of adjectives. coLangPro employs the efficient criterion (5) while coLangPro* uses the default efficiency criterion (see §6.2.2). From the viewpoint of adjectives, compared to coLangPro, coLangPro* more tends to treat adjectives as intersective. Due to these differences the provers disagree in 4 problems. Moreover, coLangPro classifies two more problems correctly than coLangPro*, e.g., FraCaS-21 is proved due to the efficient criterion (5).

Sec (Sing/All)	Single-premised (Acc %)							Multi-premised (Acc %)					Overall (Acc %)				
	BL	NL07,08	LS P/G	NLI	T14a,b	M15	**LP**	BL	LS P/G	T14a,b	M15	**LP**	BL	LS P/G	T14a,b	M15	**LP**
1 GQs (44/74)	45	84 **98**	70 89	95	80 93	82	93	57	50 80	80 **97**	73	**97**	50	62 85	80 **95**	78	**95**
2 Plur (24/33)	58	~~42~~ **75**	-	~~38~~	-	67	**75**	67	-	-	67	67	61	-	-	67	**73**
5 Adj (15/22)	40	60 80	-	**87**	-	**87**	**87**	43	-	-	29	**57**	41	-	-	68	**77**
9 Att (9/13)	67	~~56~~ 89	-	~~22~~	-	78	**100**	50	-	-	**75**	**75**	62	-	-	77	**92**
1,2,5,9 (92/142)	50	- **88**	-	-	-	78	**88**	56	-	-	66	**84**	52	-	-	74	**87**

Table 6.12: Comparison of RTE systems tested on FraCaS: LP (coLangPro), NL07 (MacCartney and Manning, 2007), NL08 (MacCartney and Manning, 2008), LS (Lewis and Steedman, 2013) with Parser and Gold syntax, NLI (Angeli and Manning, 2014), T14a (Tian et al., 2014), T14b (Dong et al., 2014) and M15 (Mineshima et al., 2015). BL is a majority (ENTAILMENT/YES) baseline. Results for non-applicable sections are strikeout.

Most Europeans who are resident in Europe can travel freely within Europe

Most Europeans can travel freely within Europe

GOLD: `neut`; FraCaS-58

In Table 6.12, the performance of LangPro is compared to the other RTE systems that have been tested on the single or multi-premised FraCaS problems. Since the FraCaS data is small and the problems are usually seen during system development, the comparison should be understood in terms of an expressive power of a system and the underlying theory. The comparison shows that the natural tableau system and LangPro improve upon the related systems and methods almost on every applicable section. Our approach succeeds in deep reasoning over both single-premised and multi-premised entailment problems. Below we briefly characterize the related approaches and systems and compare to ours.

The natural logic approach by MacCartney and Manning (2007, 2008); MacCartney (2009) models monotonicity reasoning with the exclusion relation. It also accounts for implicatives and factives. Given a premise p and a hypothesis h, the approach employs a categorial grammar style parse trees t_p and t_h as a semantic representation. A sequence of atomic edits (i.e. insertion, deletion and substitution of lexical entries) transforming p into h is used as a guide to reasoning. Then t_p is gradually transformed into t_h following the sequence. While doing so, for each atomic edit, the corresponding atomic entailment relation is calculated. The final entailment relation is obtained by joining the atomic ones. Angeli and Manning (2014) use the similar approach but model the transformation of p into h elegantly in terms of a finite-state automaton, where the transitions between states are guided by atomic edits. The latter approach does not employ parse trees that makes it more robust but imprecise.

Lewis and Steedman (2013) use first-order logic representations while combining logical and distributional semantics. The former is used for modeling function words, e.g. determiners, conjunctions and negation, while the latter models meaning of content words, e.g., verbs and nouns. Similarly to Bos et al. (2004); Bos (2008), the approach translates surface forms into first-order logic formulas. All n-ary ($n > 2$) predicates are binarized—identifying with a conjunction of binary relations between each pair of arguments. Lewis and Steedman (2013) employs distributional relation clustering for binary relations. The clustering is facilitated by tagging entities with types like `PER`, `DAT` and `LOC`. As a result the approach can map a verbal predicate $\texttt{write}\langle\texttt{PER}, \texttt{Book}\rangle$ and a relational noun

$\texttt{author}\langle\texttt{PER}, \texttt{Book}\rangle$ to the same relation cluster. Lewis and Steedman (2013) uses the FraCaS section 1 to evaluate the formal component of the system. Reasoning over first-order logic formulas is carried out with the help of the Prover9 (McCune, 2010). Taking into account that the section 1 is not representative enough for HOGQ, as we have shown above, it is not surprising that their results are decent (see LS with the parser's and gold derivations in Table 6.12).

Like the previous approach, Mineshima et al. (2015) also obtains semantic representation in the style of Bos et al. (2004) but they maintain lexical terms with higher-order semantics. For example, $\texttt{most}\big(\lambda x.\,\texttt{man}(x) \wedge \texttt{sleep}(x),\ \lambda x.\,\texttt{snore}(x)\big)$ is a semantic representation of "*every man who sleeps snores*". On the other hand, "*every*", "*a*" and "*the*" are modeled as the first-order universal ($\forall$) and existential ($\exists$) quantifiers respectively. In this way, they employ higher-order logic but most of the fragment used for natural semantics is of first-order. Their inference system CCG2λ is implemented in a proof-assistant Coq (Bertot et al., 2010), which augments the first-order inference of Coq with additional axioms and inference tactics for natural logic constructions. For instance, the upward monotonicity property of the second argument of `most` is captured with the following axiom:

$$\forall F \forall G \forall H \big(\texttt{most}(F, G) \rightarrow \big(\forall x(Gx \rightarrow Hx) \rightarrow \texttt{most}(F, H)\big)\big) \tag{14}$$

Tian et al. (2014) uses abstract denotations for linguistic semantics what are obtained from Dependency-based Compositional Semantic trees (Liang et al., 2011). The abstract denotations are formulas constructed with the operators of relational algebra. For example, $\textbf{man} \cap \pi_{\text{SUBJ}}(\textbf{sleep})$ is the abstract denotation for "*men who sleep*", where π_{SUBJ} is a projection onto domain of SUBJ. Semantics of declarative sentences are modeled via statements over abstract denotations, e.g., (15) is a statement for "*every man reads a book*", where W is a universal set containing all entities and $\times$ is the Cartesian product. The semantic representation resembles a version of description logic.[14]

$$\textbf{man} \subset \pi_{\text{SUBJ}}\big(\textbf{read} \cap (W_{\text{SUBJ}} \times \textbf{book}_{\text{OBJ}})\big) \tag{15}$$

They implement an inference engine over statements, called TIFMO, which employs axioms encoding properties of algebraic operators and lexical entries. Dong et al. (2014) further extends the engine to better account for the properties of GQs, in particular, monotonicity, conservativity and interaction with universal and existential quantifications.

The natural tableau system and LangPro differs from the above mentioned approaches and systems in several aspects. The natural logic of MacCartney and Manning (2008) heavily hinges on a sequence of edits which is in an obscure relation with formal logic. Hence, the approach cannot process multi-premised problems properly. Due to this, also some simple logical relations like de Morgan's law for quantifiers, are not captured by it: MacCartney (2009) shows that it is impossible to entail "*some birds do not fly*" from "*Not all birds fly*". It is also unclear how the sequence edit-driven approach can account for paraphrases that are obtained by swapping words rather than deleting, inserting or substituting them; consider passive alternations. On the other hand, our approach and

[14] Abstract denotations share similarity with LLFs if their operators are seen as certain lexical terms, variables or term forming operators: $\subset$ as **every**, π as the λ-abstraction, $\cap$ as the function application and W as a variable.

the prover account for these issues in a natural way. A tableau proof of the mentioned entailment is given in Figure 2.7.

The other three approaches employ formal logics and are able to reason over multiple premises. All of them, including our approach, obtain semantic representations from syntactic parse trees. The natural tableau system significantly differs from these approaches in terms of a semantic representation language. LLFs are similar to CCG derivations with non-directional categories. Hence, compared to HOL, FOL or abstract denotations, their automatic generation is simpler. Moreover, shallow reasoning on surface forms comes easy to LangPro due to resemblance of LLFs and surface forms. On the other hand, it can be thought that while logical forms come at a higher price for these approaches, their inference engines are expected to be leaner.

All in all, our system and approach differ from the above mentioned ones in their unique combination of expressiveness of high-order logic, *naturalness* of logical forms (making them easily obtainable) and flexibility of a semantic tableau method. All these properties together allow to model shallow and deep reasoning successfully in a single system.

6.4.2 Based on SICK

We evaluate both parser-specific provers and the combined LangPro on SICK-test which was held out throughout the whole learning phase. The provers use the best configuration observed in the development procedure (§6.2.2).[15]

The obtained results for all three provers are in Table 6.13a while the confusion matrix of the combined LangPro is presented in Table 6.13b. The similar results of the C&C and EasyCCG based provers show that the LLF generator was not fitted to the C&C derivations during the adaptation process (see §6.2.1). Like in the case of SICK-train, the combination of parser-specific provers gives to each prover about 1% increase in accuracy. The contribution of each prover is almost equal and there is no conflict in their positive answers (i.e. entailment vs contradiction). The accuracy of coLangPro is slightly lower than for SICK-train. This shows that the training and test portions are homogeneous. Marelli et al. (2014b) reports 84% of the inter-annotator agreement for SICK, i.e. on average major annotations represent 84% of total annotations per problem. The latter likely suggests that 84% is an approximate upper-bound for the dataset. Taking this into account, the overall accuracy of the prover is extremely high, especially give that the prover only uses WordNet relations as a knowledge base.

The combined prover confuses proofs only for two problems. In both cases, a problem encoding a contradiction was proved as entailment. Since this kind of cases are of great interest, we discuss them below. In SICK-7709, C&C and EasyCCG analyze the conclusion in a wrong way that causes the entailment relation. In particular, the second negation is under the scope of the first one. Informally speaking, the conclusion obtains the semantics equivalent to "*It is false that the girl is smiling and not wearing the glasses*". The latter semantics is entailed from "*the girl is wearing the glasses*" which itself is asserted by the premise. In other words, the problem resembles the propositional entailment $S \wedge W \vDash \neg(S \wedge \neg W)$.

[15]Notice that additionally the prover treats unknown nominal modifiers as intersective since this configuration boosts the performance on the training dataset.

SICK / Prover	test (4927 problems) Prec%	Rec%	Acc%
Baseline (majority)	-	-	56.69
ccLangPro	97.47	57.73	81.08
easyLangPro	97.47	57.73	81.08
coLangPro	97.35	60.31	**82.14**

(a) Precision, recall and accuracy of the provers obtained on SICK-test

Gold SICK-test / LangPro	Ent	Cont	Neut
Entailment	**805**	0	609
Contradiction	2	**482**	236
Neutral	26	7	**2760**

(b) Confusion matrix of coLangPro (i.e. ccLangPro+easyLangPro) on SICK-test

Table 6.13: Evaluation results of the versions of LangPro on unseen SICK-test

There is no group of $\left[\text{people}_N\ [\text{dancing}_{VP_{ng}}]_{N\backslash N}\right]_N$

A group of people are dancing

LP: E G: C SICK-3913

The blonde girl with the pink top is smiling and wearing funny glasses with a large nose attached

The blonde girl with the pink top is not smiling and not wearing funny glasses with a large nose attached

LP: E G: C SICK-7709

The reason for the entailment proof of SICK-3913 is twofold (see the proof in Figure 6.5). First, the parsers make a mistake and analyze "*people dancing*" as a constituent in the premise. From a semantics point of view, this mistake is not crucial but it is decisive for the syntax. Due to this mistake, it is not possible to align terms corresponding to "*a group of people*", otherwise the proof for contradiction is straightforward. Second, the rigidity of syntactic types—an NP in a PP takes scope over the NP to which PP is attached. In this way, "*people*" has wide scope in the LLFs obtained from the CCG derivations. This allows the premise to introduce an entity c for a person that dances. Taking the entity c into account, the conclusion and ($\exists$GRP) infer that c is not dancing. This inference is licensed by the intuition behind ($\exists$GRP)—whatever is case for "*a group of* A", the same is for A. On the side of the prover, we could prevent this kind of false proofs by having more flexible scope ordering in LLFs. The latter is left for future research.

In Table 6.14, we compare the results of LangPro to other RTE systems that were evaluated on SICK. In Table 6.14a we present the results of several (top) RTE systems from the SemEval-14 task[16] (Marelli et al., 2014a). The median accuracy in the task was 77.1% and only five systems were able to overcome the accuracy threshold of 80%. Along with these top systems, we also include UTexas (Beltagy et al., 2014) due to its logic-based nature. In Table 6.14b, there are several relevant systems that were evaluated on SICK outside the SemEval task. For each system in Table 6.14a, we give its precision, recall and accuracy. To show the impact of our prover on other systems in terms of recall, we present the improvement (+LP) it gives to a system. In this experiment, a system blindly adopts the entailment and contradiction judgments of LangPro.[17]

[16]The full name of the task is *Evaluation of Compositional Distributional Semantic Models on Full Sentences through Semantic Relatedness and Textual Entailment*. More details about the task, including the dataset and evaluation results, are available at http://alt.qcri.org/semeval2014/task1/

[17]The improvement scores are presented only for the systems participated in SemEval-14 as their

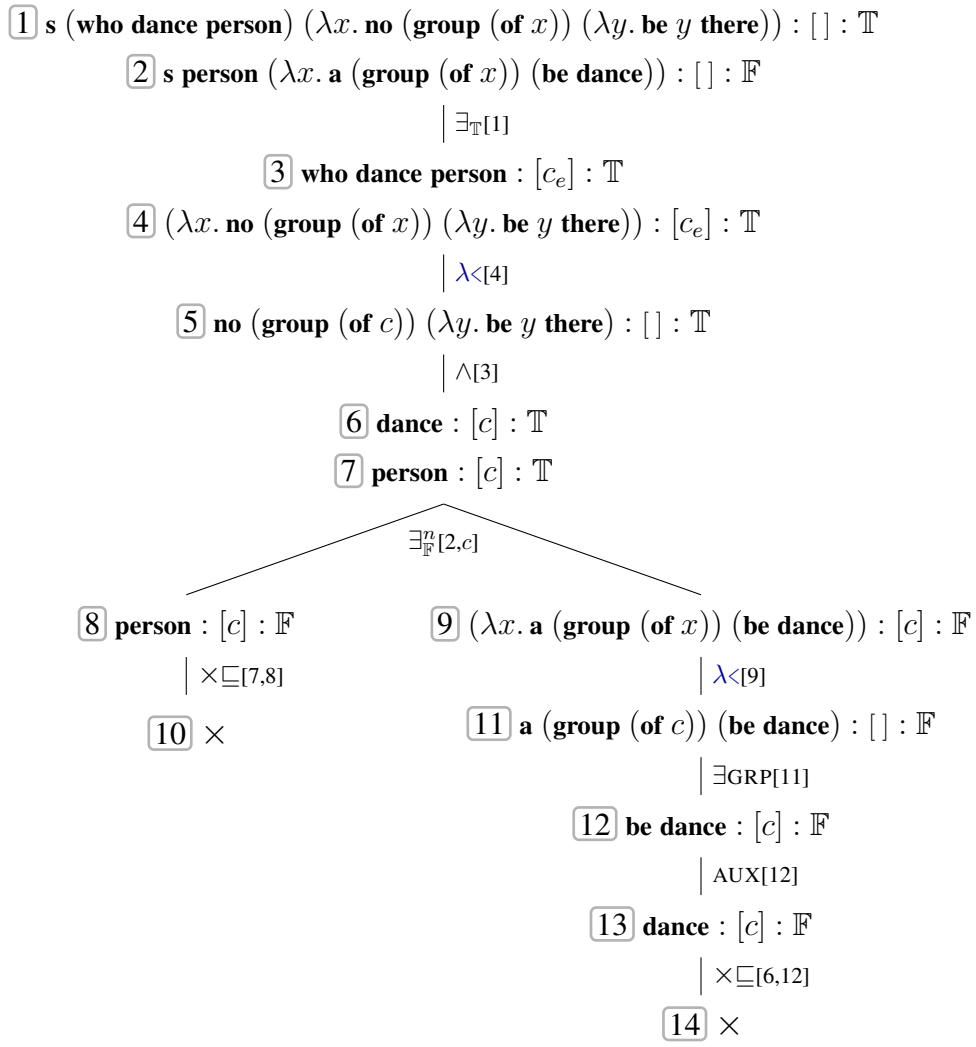


Figure 6.5: A tableau wrongly proves SICK-3913 as entailment instead of contradiction. The LLFs are obtained from the C&C derivations. LLFgen models "*people*" as **s person**.

The four top systems at the SemEval task employ machine learning classifiers to judge a pair of sentences for an entailment relation. Illinois-LH (Lai and Hockenmaier, 2014) associates each problem with 10 features (e.g., related to negation, antonyms, alignment, distributional and denotational similarities) and classifies a set of features with the Maximum Entropy classifier. ECNU (Zhao et al., 2014) extracts 72 features (involving semantic and surface text similarities, text difference measures, corpus based features, etc.) for each premise-conclusion pair and based on them a problem is classified with support vector machine (SVM) method. UNAL-NLP (Jimenez et al., 2014) uses Soft Cardinality for additional features and decision tree with grafting for classification. SemantiKlue (Proisl et al., 2014) employs 39 features measuring semantic similarity between two texts, where a word-to-word alignment plays crucial role. The set of features are then classified by SVM. Arguably these approaches do not intend to model human reasoning but to identify

runs are available online at http://alt.qcri.org/semeval2014/task1/index.php?id=data-and-tools

SemEval-14 systems	Prec%	Rec%	Acc%	(+LP)	NWS%
Baseline (majority)	-	-	56.69		39.7
Illinois-LH	81.56	**81.87**	**84.57**	(+0.65)	72.8
ECNU	84.37	74.37	83.64	(+1.77)	72.7
UNAL-NLP	81.99	76.80	83.05	(+1.48)	71.2
SemantiKLUE	85.40	69.63	82.32	(+2.84)	71.5
The Meaning Factory	93.63	60.64	81.59	(+2.78)	73.0
UTexas (Prob-FOL)	**97.87**	38.71	73.23	(+9.44)	62.5
LangPro	**97.35**	60.31	82.14		**74.8**

(a) Comparison of LangPro to the top five systems of the SemEval-14 task on SICK-test. Beside the standard measures, we also present the increase in accuracy (+LP) LangPro gives to each system. NWS is a normalized weighted score which is calculated according to the weights presented in Table 6.14c

RTE systems	Acc%
Prob-FOL	76.52
Prob-FOL*+Rules	**85.10**
Nutcracker+PPDB	79.60
ABCNN-3	**86.20**
LSTM RNN+SNLI	80.80

(b) Other related systems evaluated on SICK-test

Gold\System	E	C	N
Entailment	2	-2	0
Contradiction	-2	2	0
Neutral	-1	-1	1

(c) Weights for each type of classification

Table 6.14: Comparing LangPro to other RTE systems tested on SICK-test

properties that are successful in recognition of textual entailments. For example, one of the regularities Lai and Hockenmaier (2014) report is that 86% of the contradiction problems are recognizable based on presence of a negative word in the sentences. After we add such an ad-hoc (and trivial) classifier for negative words to LangPro, the accuracy reaches 83%.

The Meaning Factory (Bjerva et al., 2014) is mainly based on Nutcracker. The latter dates back to Bos and Markert (2005). Nutcracker employs the wide-coverage semantic processing tool Boxer (Bos, 2008), in combination with the C&C tools, and first produces Discourse Representation Structures of DRT (Kamp and Reyle, 1993) and then translates them into FOL (Curran et al., 2007). Reasoning over FOL formulas is carried out using off-the-shelf FOL theorem provers and model builders. Bjerva et al. (2014) reports that Nutcracker (with WordNet by default) achieves accuracy of 77.6% on SICK-test. Adoption of the paraphrases from PPDB (Ganitkevitch et al., 2013) gives the system 2% boost in accuracy. Finally, the Meaning Factory represents augmentation of Nutcracker with an SVM classifier. The classifier tries to overcome the problem of low recall; it reclassifies the problems that were judged by Nutcracker as neutral. For this, the classifier takes a relatedness score as an input from the semantic similarity system—the second system presented in Bjerva et al. (2014).

The RTE system UTexas (Beltagy et al., 2014) also uses FOL representations obtained from Boxer but compared to Nutcracker it employs probabilistic FOL with Markov Logic Networks (MLN). Prob-FOL (Beltagy and Erk, 2015) and Prob-FOL+Rules (Beltagy et al., 2015) are more advanced versions of the same RTE system. The latest version of the system (Beltagy et al., 2015) solves an RTE problem $\langle T, H \rangle$ as follows. Like our approach, it employs both C&C and EasyCCG parsers to obtain First-Order Logic (FOL) formulas t and h through Boxer. Reasoning over the formulas amounts to calculate conditional probabilities $P(h|t, KB, W_{t,h})$ and $P(\neg h|t, KB, W_{t,h})$, where KB is a knowledge base and $W_{t,h}$ is the world configuration—it depends on t and h and lists constants and prior probabilities of atomic formulas. These numbers are then mapped to a final entailment relation by an SVM classifier. A main reason for the high results Beltagy et al. (2015) obtain on SICK is their KB construction. In addition to WordNet, PPDB and a

few hand-coded rules, the KB contains automatically extracted, classified and weighted rules from the dataset. The rules are extracted from SICK-train using a modified version of Robinson resolution for FOL. Then a classifier is trained over these extracted rules in order to identify such rule as entailment, contradiction or neutral. The extracted rules are often long and represent *almost solutions* for the SICK problems.[18] For example, if we consider the false neural problems in Table 6.10, their KB directly provides the key relations for the problems, where numbers denote the position of a word in the corresponding SICK-problem:

"*man-2 woman-5*" → "*two-1 persons-2*"	for SICK-4275
"*container-6 made-7 of-8 plastic-9*" → "*plastic-6 container-7*"	for SICK-4553
"*hedgehog-5*" → "*small-5 animal-6*"	for SICK-4974

The methods using neural networks have also been successful on the dataset. Bowman et al. (2015a) achieve their best results with the Long Short-Term Memory Recurrent Neural Network (LSTM RNN). The model is first trained on the Stanford Natural Language Inference (SNLI) corpus, containing more than 500K RTE problems, and then on SICK-train. Recently, Attention Based Convolutional Neural Network (ABCNN) of Yin et al. (2015) achieved the highest score on the dataset. In addition to word embeddings, their model employs features related to word overlaps and a presence of negation words.

The performance of LangPro qualitatively differs from the results obtained at the SemEval-14 task. Our prover turns out to be extremely reliable with its proofs. It demonstrates almost perfect precision (>97%) while being competitively accurate. This is achieved while using only WordNet for the KB, in contrast to other systems employing several resources. When compared to the Meaning Factory, the system with high precision, LangPro allows three times less false positives—classifying neutral problems as entailment or contradiction. To show this difference with the other systems, we suggest a normalized weighted score (NWS) which takes into account bonuses and penalties for true positives and false positives respectively. The weights for each type of classification is given in Table 6.14c.[19] For example, according to the weights, it is better to classify a contradiction problem as neutral rather than as entailment. Weighted scores are normalized by the maximum score, which is obtained by the gold standard. Based on the NWS, our prover gets the highest results as it makes the least errors in conjunction with high accuracy.

From a practical point of view, it is interesting to see where the reasoning with the natural tableau system outperforms other approaches. In Table 6.15, we give some of those problems fromSICK-test that are correctly proved by LangPro but, on the other hand, are wrongly classified by all the top five systems of the SemEval task. Among other problems concerning Booleans, e.g., SICK-247, and verb sub-categorization, e.g., SICK-3527, 3570, surprisingly there are also some simple problems too, like SICK-2895, 3806 and 4479.

Our approach significantly differs from most of the above presented methods as we employ logical forms for reasoning rather than a bunch of features extracted from an en-

[18]The rules are available at https://github.com/ibeltagy/rrr

[19]The presented weights reflects a simple intuition and is not based on some empirical evidence or experiment. Moreover, we could tweak the weights by taking into account the distribution of gold labels in the dataset. We owe the idea of using weighted score for comparison to Jan Sprenger.

ID	G	Premise	Conclusion
247	C	P: The woman is not wearing glasses or a headdress C: A woman is wearing an Egyptian headdress	
406	E	P: A group of scouts are hiking through the grass C: People are walking	
2895	C	The man isn't lifting weights	The man is lifting barbells
3527	E	P: A person is jotting something with a pencil C: A person is writing	
3570	C	The piece of paper is not being cut	Paper is being cut with scissors
3608	N	P: A monkey is riding a bike C: A bike is being ridden over a monkey	
3806	E	A man in a hat is playing a harp	A man is playing an instrument
4479	E	The boy is playing the piano	The boy is playing a musical instrument

Table 6.15: The problems from SICK-test that were proved correctly by both ccLangPro and easyLangPro but failed by all the top five systems at the SemEval-14 task.

tailment problem. Many of those approaches are also crippled for reasoning over multiple premises. Compared to Nutcracker and Prob-FOL, LangPro and the underlying natural tableau system differ from them in three main aspects:

(i) Our approach contributes to the project of natural logic, which means that reasoning is carried out on logical forms that resemble surface forms and contain elements of natural language syntax. As a result, obtaining such logical forms from linguistic expressions is cheaper compared to translation into FOL.

(ii) The underling higher-order logic of LLFs is more expressive than FOL. For example, it can model GQs and subsective adjectives in a straightforward way.

(iii) Our system and prover are based on a semantic tableau method which represents an intuitive search for a situation satisfying a set of formulas. The tableau system can be seen as a model builder and a prover at the same time. Due to these properties, our approach is self-contained and does not appeal to other systems for reasoning.

On the other hand, the two approaches have their own merits. They both obtain FOL formulas from Boxer, which means that the pronouns in the formulas are already resolved with the help of Discourse Representation Theory. While Prob-FOL can easily integrate distributional semantics and produce fuzzy judgments, Nutcracker does not need to worry about automated reasoning as there are several off-the-shelf theorem provers available for FOL and the separate community is lively working on that issue.

We have presented the evaluation results of the prover on the FraCaS and SICK datasets. The natural tableau prover is successful in deep reasoning over multi-premised problems and in shallow reasoning for relatively short sentences. The results demonstrate high accuracy with almost perfect precision meaning that if the prover finds a proof there is less than 3% chance that it is wrong. The high precision is explained by the set of sound inference rules. On the other hand, high recall—unusual for rule-based systems—is explained by the ability of shallow reasoning and the inventory of rules that covers ample amount of syntactic constructions and semantic phenomena.

6.5 Conclusion

In the chapter we have described the learning phase that employs the FraCaS and SICK datasets. After the phase, LangPro has been evaluated against the certain sections of FraCaS and the unseen test portion of SICK. Both evaluation results are competitive with an almost perfect precision. The prover rarely finds false proofs and sometimes false proofs indicate noisy gold labels. On FraCaS, the prover demonstrates state-of-the-art semantic competence while on SICK there is less than 2% gap between performances of human and LangPro.

The FraCaS sections such as anaphora, ellipsis, comparatives and temporal reference were excluded since currently we do not account for these phenomena in the natural tableau system. In contrast to other RTE systems, LangPro have not learned much from the training portion of SICK. We used this portion only for development purposes—mainly measuring the contribution a certain parameter makes to the performance. The reason for this is that the adaptation procedure is expensive. At this moment, it requires assistance of an expert to teach the prover how to solve certain entailment problems.[20] Despite the high price of learning, learned tableau and term fixing rules are reusable for future applications.

We would like to emphasize that the high results on the datasets were achieved with a very simple pipeline: a parser component, a post-processor of parse trees and an inference engine backed up with WordNet as a KB. This fact has two direct implications. First, it shows a high reasoning capacity of the natural tableau system. Neither use of a CCG parser for wide-coverage semantics Bos et al. (2004) nor WordNet as a lexical knowledge base are novel. But the tableau system for natural logic is the only new component in this intuitive and simple pipeline. So, it gives evidence that the success of the simple pipeline lies in the natural tableau system. Second, we showed that with a powerful reasoning engine it is possible to achieve high results in the RTE task with shallow semantic representations, such as LLFs, and with a lexical database, such as WordNet. Our experiment on the SICK showed that WordNet relations (with several additional relations specific to the dataset) provide sufficient knowledge for achieving competitive results on SICK.

It is hard to scale up a purely logic-based approach to wide-coverage reasoning, and many could be skeptical about the current approach and its wide-coverage nature. But we would like to stress the hybrid reasoning skills of our approach. On one hand, the natural tableau system carries out shallow reasoning with the help of monotonicity reasoning and tableau rules that operates on common syntactic constructions. On the other hand, it is also capable of deep reasoning via the rules that infer semantic terms from syntactic ones, i.e. shifting from shallow structures to deep structures. In fact, the high results on the FraCaS sections and the SICK dataset demonstrate success of our approach in deep and shallow reasoning, respectively. Moreover, we are not aware of a wide-coverage model for natural language inference that combines shallow and deep logical reasoning on the same level. It is also an interesting fact that before LangPro, no RTE systems were evaluated on both FraCaS and SICK datasets.

[20] In order to overcome this shortcoming to some extent, as future work we suggest automatic lexical knowledge acquisition from data (§7.2.2).

Appendix E

Algorithm 4: A human-assisted adaptation procedure of LangPro to a textual entailment problem. The underlined expressions and commands are determined and carried out respectively by an expert.

Input: A list of premises P, a hypothesis h and a *Gold* answer
Output: A Boolean value indicating whether the problem is classified correctly

```
1  try
2      ProAnswer ← ccLangPro(P, h)        // LangPro/2 (see algorithm 3) with C&C
3  exception                              // If an exception is raised by LangPro/2
4      case "Entailment & contradiction"  // Indeterminacy case
5          go to line 11;
6      otherwise   // Involves the case when a parser does a mistake; see algorithm 3
7          return FALSE
8  if Gold = ProAnswer then
9      return TRUE                        // LangPro guesses the gold answer
10 else
11     switch Reason do                   // Explore the reasons for misclassification
12         case at least one of LLFs is not correct
13             if FixTerm/1 can be improved then   // See algorithm 3 for FixTerm/1
14                 Improve FixTerm/1
15                 go to line 1
16             else
17                 return FALSE
18         case A semantic relation is missing in the KB
19             Add the missing relation in the KB
20             go to line 1
21         case A rule is missing in the IR
22             Add the missing rule in the IR
23             go to line 1
24         case An information is missing in the signature
25             Add the missing information in the signature
26             go to line 1
27         otherwise
28             return FALSE
```

Chapter 7

Conclusion

The chapter concludes the thesis by summarizing its contributions and sketching several directions of future work. First, we list three main contributions of the thesis: extension of the natural tableau system, automatic generation of logical forms and automated theorem prover for natural language. Then several directions of future work are discussed, where each direction is provided with a short description or anticipated issues. The described future work covers lexical knowledge acquisition, a possible combination with distributional semantics, employing additional textual entailment datasets and obtaining the logical forms from dependency trees. The chapter ends with final remarks.

7.1 Summing up

In the thesis, we have presented the natural tableau system for a version of natural logic, and based on it a tableau theorem prover for natural language, called LangPro, has been implemented. The natural tableau system and the theorem prover prove natural language arguments by refuting them: they search a counterexample for the argument, i.e., a situation which makes the premises true but the conclusion false. The prover demonstrates competitive results and high semantic competence on specific textual entailment datasets. The employed formal natural logic (Muskens, 2010) is higher-order type logic. The terms of the logic, called Lambda Logical Forms (LLFs), represent logical forms of linguistic expressions, and they resemble linguistic surface forms. Below we summarize the thesis by listing its three main contributions.

We have extended the tableau system of (Muskens, 2010) in order to make it robust and suitable for reasoning over wide-coverage natural language text. The extension is carried out in three directions: (i) integrating syntactic types in the type system; (ii) introducing an extra slot in the format of tableau entries which facilitates accommodation of event semantic and captures a link between remote modifiers, e.g., adverbial phrases, and their heads; and finally (iii) the collection of tableau rules that enable reasoning over open-domain text. The inventory of tableau rules counts around 80 rules. Most of the rules, apart from the rules found in the initial tableau system (around 25), are designed manually in a data-driven fashion. We employed the (portions of) RTE datasets and the LangPro theorem prover to facilitate search of tableau rules. The inventory consists of two categories of rules concerning formal and linguistic elements. In particular, the for-

mal elements involve Booleans, monotonic operators, argument and modifier lists, events, semantic inclusion, exclusion and exhaustion, (anti-) additive operators and projectivity properties. On the other hand, the linguistic elements cover adjectives, prepositional phrases, passive constructions, verb subcategorization, definite descriptions, open compound nouns, light verb constructions, attitude verbs, expletives and copula.

Theorem proving over wide-coverage natural language sentences is impossible without their logical forms. For this reason, we designed and implement an LLF generator which produces logical forms from Combinatory Categorial Grammar (CCG) (Steedman, 2000) derivations. First, CCG derivations for linguistic expressions are obtained from the state-of-the-art CCG parsers: C&C (Clark and Curran, 2004b, 2007) and EasyCCG (Lewis and Steedman, 2014a). Then the derivations are transformed into λ-term alike structures, called CCG terms. The latter terms are made further similar to λ-terms and more semantically adequate by eliminating type-changing rules and correcting systematic mistakes of the parsers via collected fixing rules, respectively. The final stage in the generation of LLFs is to type-raise quantified noun phrases in a corrected term and to treat them as generalized quantifiers (GQs).

An implemented tableau theorem prover for the natural logic, called NLogPro, is faithful to the developed natural tableau system. While building a tableau proof, NLogPro maintains a list of branches and an optional tableau tree. The latter structure facilitates reading a flow of tableau proofs. In addition to the collected tableau rules, the prover employs derivable rules—the rules that represent shortcuts for several rule applications. The use of the derivable rules decreases size of tableau proofs and makes it easier to read the proofs. Combining NLogPro with the LLF generator results in a tableau theorem prover for natural language, LangPro. The prover demonstrates state-of-the-art semantic competence on the certain sections of the FraCaS dataset (Cooper et al., 1996). Using only WordNet Miller (1995) for the Knowledge Base (KB), LangPro obtains competitive results on an unseen portion of SICK Marelli et al. (2014b). This fact shows that with a purely rule-based reasoning and a default lexical database, such as WodNet, it is possible to achieve the high results comparable to state-of-the-art RTE systems, which employ machine learning techniques combined with lexical, phrasal or image databases and statistics from language corpora.

The thesis contributes to the research fields of computational semantics and natural language processing (NLP) as it tackles the problem of the latter with the methods from the former. In particular, we employ a formal logic for linguistic semantics and theorem proving for wide-coverage reasoning. In this way, our work supports the research line that scales up formal semantics for NLP applications. Furthermore, our theorem prover represents a novel application of natural logic that is backed up with shallow and formal logical reasoning at the same time. Before our work, natural logic in NLP community was understood as a mode of shallow reasoning which is based on phrase substitutions and is limited to single-premised arguments.

7.2 Future work

A theory that attempts to account for an immense problem such as reasoning in natural language unsurprisingly has much of future work and applications. Below we mention future work on the natural tableau that focuses on the four aspects concerning robustness and extension of the approach. In particular, first we discuss applying our natural tableau system and theorem prover to other (recognizing textual entailment) RTE datasets that contain problems of a different kind than FraCas and SICK do. Then we describe an automatic way of acquiring lexical knowledge from RTE datasets with the help of the theorem prover. A possible architecture for the prover is presented that employs a classifier for lexical knowledge and incorporates distributional semantics. In the end, we give an example of obtaining LLFs from widely used structures such as dependency trees.

7.2.1 Trying other RTE datasets

Up to now we have used the FraCaS (Cooper et al., 1996) and SICK (Marelli et al., 2014b) datasets for developing and evaluating the natural tableau system and the theorem prover. Although the datasets are collected with different goals, they still share similarities: they contain short sentences (on average 10 words per sentence) with a relatively simple structure, the sentences are artificially constructed and the datasets require no world, i.e. encyclopedic, knowledge (e.g., Tbilisi is a capital of Georgia).

On the other hand, there are other RTE datasets which differ from the above two and are constructed with different purposes. Since the first RTE challenge Dagan et al. (2006), series of RTE datasets were collected based on newswire text (see RTE1-1877). These datasets are considered hard from different perspectives: their problems assume world knowledge and contain complex sentences that are difficult for syntactic parsing. The Stanford Natural Language Inference (SNLI) corpus[1] (Bowman et al., 2015a) is initially constricted to enable "a neural network-based model to perform competitively on natural language inference benchmarks". The corpus consists of 570K problems and its sentences are comparable with those of SICK (see SNLI-1). However, its sentences are manually constricted using crowdsourcing and intuition behind the CONTRADICTION label is different from the standard one.[2]

The English team arrived last night in Lisbon, Portugal, to play its first Euro 2004's match
Euro 2004 is held in Portugal
GOLD: ent; RTE1-1877

Two men on bicycles competing in a race
A few people are catching fish
GOLD: cont; SNLI-1

[1] http://nlp.stanford.edu/projects/snli/

[2] The labeling assumes that sentences in a problem are alternative (but possibly distorted) captions of some hidden image. This primes annotators that the events in the sentences co-refer. Thus, a pair in SNLI-1, with the original ID 6170429974.jpg#3r1c, is judged as CONTRADICTION since fishing and cycling events cannot co-refer.

Adapting and evaluating the tableau system and the prover to these datasets will give insight into two directions. How flexible and robust is the natural tableau theory? And how the performance of the theorem prover on these datasets compares to the results of other RTE systems? Concerning the datasets of RTE challenges, three major challenges are expected: (i) explaining a wide-range of type-changing rules of the CCG parsers, (ii) designing tableau rules that surmounts parsing errors to some extent, and (iii) acquiring world knowledge either automatically from the datasets or from knowledge resources. In case of the SNLI corpus, we need to account for its specific notion of contradiction. For example, this can be reached by a tableau rule that enforces co-reference of the events found in an entailment problem.

7.2.2 Acquisition of lexical knowledge

Knowledge acquisition is considered as one of major bottlenecks for RTE systems (Dagan et al., 2013, p. 7). One can employ off-the-shelf knowledge resources, e.g., WordNet (Miller, 1995), FrameNet (Baker et al., 1998), VerbNet (Schuler, 2005) or Cyc (Lenat and Guha, 1989), but they are never enough. The common practice in NLP is that first an RTE system automatically acquires knowledge from a training RTE data and then the trained system is evaluated on an unseen test data, of the same kind as the training data.

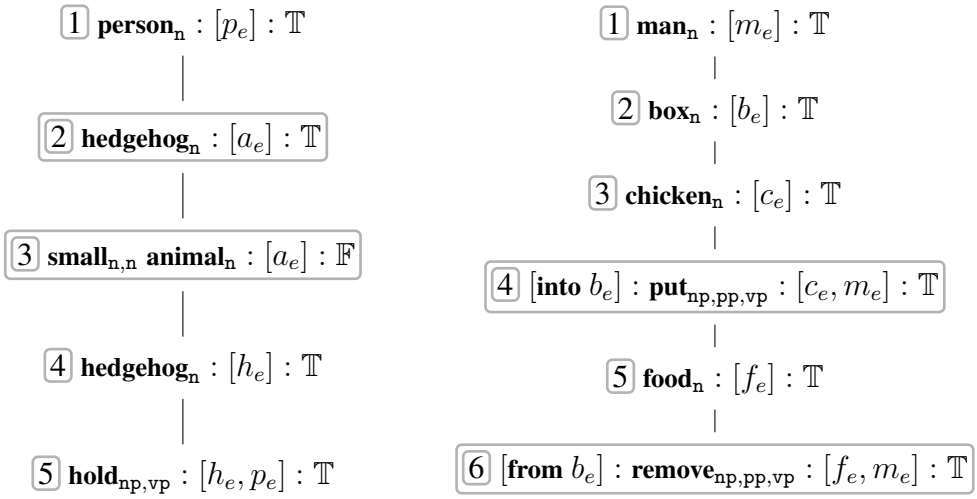
One way to acquire lexical (and possibly world) knowledge with the help of the natural tableau theorem proving is to use abduction. Given an RTE problem and its gold label, first we construct a candidate tableau, i.e. the tableau that is responsible for a correct guess. Then we find a semantic relation that would close the tableau if we had it in the KB. For example, consider the false neutral SICK-4974 from § 6.3.3, which is not proved due to a lack of **hedgehog** $\sqsubseteq$ **small animal** in the KB. Let us call a node *lexical* if its LLF consists of a lexical head term and optional lexical terms modifying the head. Now, consider all lexical nodes (see Figure 7.1a) that are shared by every open branch of the candidate tableau of SICK-4974. Put differently, we take all lexical nodes that are found in the intersection of all open branches. Taking into account the set of closure rules of the tableau system, the nodes 2 and 3 are the only lexical ones that can close the tableau, and it is done in combination with **hedgehog** $\sqsubseteq$ **small animal**, which is actually the missing lexical knowledge.

Someone is holding a hedgehog
Someone is holding a small animal
GOLD: ent; SICK-4974

A man is removing some food from a box
The man is putting chicken into the container
GOLD: cont; SICK-5397

Skeptics might consider the knowledge acquisition from SICK-4974 trivial, which can also be carried out with a simple alignment of phrases. Our suggested method also successfully leads to the non-trivial knowledge **put** | **remove**[3] required by SICK-5397: in Figure 7.1b, inconsistency of the nodes 4 and 6 is the only evidence that leads to a contradiction proof for the problem.

[3] **put into** | **remove from** is another alternative relation and the choice between them depends on particular details.



(a) Extracted nodes for SICK-4974 (b) Extracted nodes for SICK-5397

Figure 7.1: Examples of lexical knowledge acquisition from branches. The framed nodes are clues for the missing knowledge **hedgehog** $\sqsubseteq$ **small animal** and **put into** $\mid$ **remove from**.

Abduction, known as an inference to the best explanation, is often used with weights or scores in semantic applications in order to select the *best* knowledge, axiom or assumption (Hobbs et al., 1993; Fowler et al., 2005; Raina et al., 2005). In the described abductive knowledge leaning, the length of a term can be seen as a primary indicator for the best lexical knowledge. We believe that the suggested learning method will significantly boost the performance of LangPro on RTE datasets such as SICK.

7.2.3 Pairing with distributional semantics

Distributional semantics, i.e. vector space semantics (Turney and Pantel, 2010; Erk, 2012), can be easily integrated in the natural tableau system for modeling lexical semantics. The tableau rules are good at breaking down complex phrases into smaller ones and modeling semantics of functional words such as quantifiers, negation, Boolean connectives and prepositions. On the other hand, distributional semantics are robust on lexical level when concerning open class words. In this way, we can combine and complement these two approaches.[4]

The combination we have in mind is the following. Let us assume a function or a fuzzy predicate Rel which assigns a weight to a concrete relation $p_1 R p_2$, where p_1 and p_2 are short terms and R is one of the three semantic relations $\mid, \sqsubseteq, \smile$. The function Rel will work on a lexical level and provide closure rules with lexical semantic relations at a certain confidence level. For example, a high weight for **hedgehog** $\sqsubseteq$ **small animal** will render the nodes 2 and 3 in Figure 7.1a as inconsistent and will lead to the branch closure. Similarly, a high weight for **put** $\mid$ **remove** will identify the consequences of 4 and 6 in Figure 7.1b as inconsistent. In the end, a closed tableau can be considered as a

[4] A similar hybrid approach has already been applied by Lewis and Steedman (2013). They combine first-order logic semantics and distributional clustering at the level of predicate-argument structure. See §6.4.1 for more comparison of their work to ours.

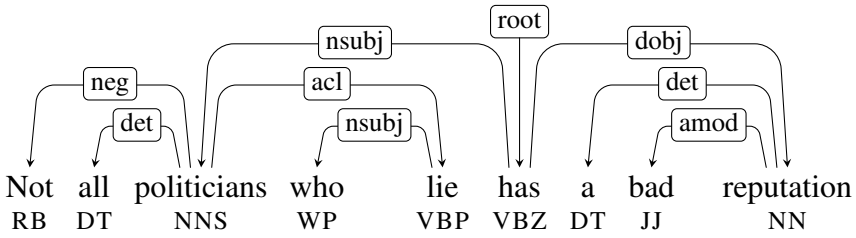


Figure 7.2: A dependency tree represented as a directed acyclic graph. Arcs are going from a head word to a dependent word. Each arc expresses a certain type of dependency. A verb is a root of a sentential dependency tree.

proof if all the weights of the employed relations or their combination is above a certain threshold. Weighted relations can also be used to classify entailment problems with a confidence score.

Though we are not aware of any implementation of the function Rel as such, there are several works in this direction that attempt to model the $\sqsubseteq$ and $|$ semantic relations using distributional semantics. Kruszewski and Baroni (2015) tried to model (in)compatible words while the others including Kotlerman et al. (2010); Levy et al. (2015); Shwartz et al. (2016) focused on the hypernymy/entailment relation.

7.2.4 Generate LLFs from dependency trees

In Chapter 3, we generated LLFs from the CCG (Steedman, 2000) derivations produced by the CCG parsers C&C (Clark and Curran, 2007) and EasyCCG (Lewis and Steedman, 2014a). The CCG framework was chosen as it has a transparent syntax-semantic interface. Despite this advantage of CCG, we could opt for a different grammar formalism, for instance, Dependency Grammar (DG) (Tesnière, 1959), and generate LLFs from dependency trees (see Figure 7.2). Dependency parsing is well-studied and established branch in NLP (Kubler et al., 2009; Nivre, 2010; Nivre et al., 2016). There are several fast and accurate parsers[5] and a bunch of treebanks (Nivre et al., 2016)[6] available for DG. Taking into account advances in dependency parsing, employing dependency trees as a source can significantly improve the quality of LLFs.

In order to obtain LLFs from dependency trees, one needs to explain dependencies in terms of term composition. It is not always the case that a head is a function and the dependents its arguments: see the `nsubj`, `acl` or `det` dependencies in Figure 7.2. For the demonstration, we define a simple recursive function $\texttt{Term}/1$ which transforms the dependency tree of Figure 7.2 into a CCG term. The function is defined in terms of the transformation rules in (R1–R7), and they explain dependencies involved in the running tree. The rules are sorted according to priority: during application, a rule above has a priority over the below ones. We denote a dependency tree as a *headed multiset* $\langle H \mid \overrightarrow{c} \rangle$, where a head H is a root and a tail $\overrightarrow{c}$ is a multiset of arc-tree $a_i T_i$ pairs such that a_i goes from H to the root of T_i. The transformation procedure starts from the initial dependency

[5] McDonald et al. (2005); Nivre et al. (2007); Martins et al. (2013); Manning et al. (2014) inter alia.

[6] A collection of multilingual treebanks formatted in the cross-lingual dependency style Universal Dependencies: http://universaldependencies.org

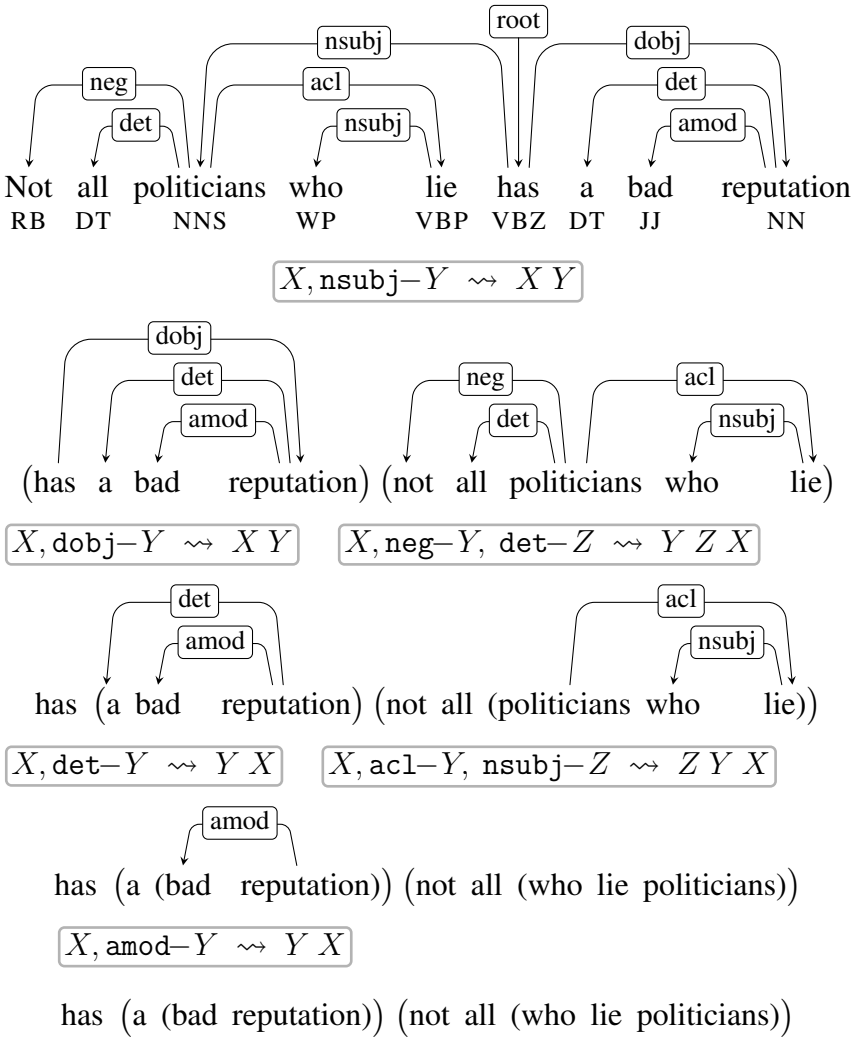


Figure 7.3: The step by step transformation of a dependency tree into a CCG term. Each step is accompanied with a simple schematic rule corresponding to a transformation rule.

tree and processes its sub-trees. The rules are applied according to their order, e.g., `nsubj` is explained earlier than `dobj` as (R1) has higher priority than (R2).

$$\texttt{Term}\langle X \mid \vec{\alpha}, \texttt{nsubj}{-}Y\rangle \rightsquigarrow \texttt{Term}\langle X \mid \vec{\alpha}\rangle\ \texttt{Term}(Y) \tag{R1}$$
$$\texttt{Term}\langle X \mid \vec{\alpha}, \texttt{dobj}{-}Y\rangle \rightsquigarrow \texttt{Term}\langle X \mid \vec{\alpha}\rangle\ \texttt{Term}(Y) \tag{R2}$$
$$\texttt{Term}\langle X \mid \vec{\alpha}, \texttt{neg}{-}Y,\ \texttt{det}{-}Z\rangle \rightsquigarrow \texttt{Term}(Y)\ \texttt{Term}(Z)\ \texttt{Term}\langle X \mid \vec{\alpha}\rangle \tag{R3}$$
$$\texttt{Term}\langle X \mid \vec{\alpha}, \texttt{det}{-}Y\rangle \rightsquigarrow \texttt{Term}(Y)\ \texttt{Term}\langle X \mid \vec{\alpha}\rangle \tag{R4}$$
$$\texttt{Term}\langle X \mid \vec{\alpha}, \texttt{amod}{-}Y\rangle \rightsquigarrow \texttt{Term}(Y)\ \texttt{Term}\langle X \mid \vec{\alpha}\rangle \tag{R5}$$
$$\texttt{Term}\langle X \mid \vec{\alpha}, \texttt{acl}{-}Y,\ \texttt{nsubj}{-}Z\rangle \rightsquigarrow \texttt{Term}(Z)\ \texttt{Term}(Y)\ \texttt{Term}\langle X \mid \vec{\alpha}\rangle \tag{R6}$$
$$\texttt{Term}\langle X, [\,]\rangle \rightsquigarrow X \tag{R7}$$

The whole transformation procedure is presented in Figure 7.3. The transformation terminates when all terminal nodes are reached. In the end of the procedure, we get a CCG term. The types of lexical elements can be obtained by (partially) encoding them in

the transformation rules and using POS tags. Checking the obtained term on typing can be used to induce the rest of uncovered lexical types. After recovering the types, then it is possible to obtain corresponding LLFs by type-raising quantified NPs (§3.5).

In addition to the above presented future work, there are several directions that deserve to be mentioned. One of them is to model anaphoric expressions in the natural tableau system. Taking into account that tableau branches correspond to possible situations, one might think of a tableau rule that applies to an entry with an anaphoric term and accommodates it in (the situation of) the branch. Accounting for textual entailments with comparatives is also a possible future work. The current work can borrow ideas from a syllogistic logic with comparative adjectives (Moss, 2011). The last possible direction is to employ LLFs without type-raised quantified NPs. It is motivated by the fact that syntactic types in LLFs cannot give certain kinds of scope ordering (see §3.5). This means that CCG terms (§3.4) will become logical forms and they will have underspecified semantics with respect to scope ambiguity. After all such underspecification in logical forms is in accordance with natural logic.

7.3 Final remarks

In the thesis, we have extended the analytic tableau system of natural logic (Muskens, 2010) in order to make it suitable for wide-coverage natural reasoning. The tableau theorem prover based on this system was implemented and adapted to the textual entailment datasets that require both shallow and deep reasoning. As a result the prover operates on the logical forms, which resemble syntactic structures, and is able to reason over multiple premises involving various syntactic or semantic phenomena. High results obtained on the datasets indicate success of our methodology. Moreover, our approach puts natural logic and a formal logic together in one system in order to reason over wide range of linguistic expressions. This hybrid combination is novel for the NLP community as before natural logic was understood as an inference device that is driven by word substitutions and is restricted to single premises entailment problems. We hope that our work will serve as a motivation for scaling up existing natural logic calculi and will stimulate applications of higher-order logic to textual entailment.

Acronyms

ACG Abstract Categorial Grammar.

CCG Combinatory Categorial Grammar.

CDSM Compositional Distributional Semantic Models.

DG Dependency Grammar.

DRS Discourse Representation Structure.

DRT Discourse Representation Theory.

FOL First-Order Logic.

GQ Generalized Quantifier.

KB Knowledge Base.

LLF Lambda Logical Form.

MWE Multiword Expression.

NLI Natural Language Inference.

NLP Natural Language Processing.

NP Noun Phrase.

POS Part of Speech.

PP Prepositional Phrase.

PV Prepositional Verb.

PVC Particle-Verb Construction.

QA Question Answering.

RAL Rule Application Limit.

RTE Recognizing Textual Entailment.

VP Verb Phrase.

WSD Word Sense Disambiguation.

Bibliography

Abzianidze, L. (2015a). A pure logic-based approach to natural reasoning. In Brochhagen, T., Roelofsen, F., and Theiler, N., editors, *Proceedings of the 20th Amsterdam Colloquium*, pages 40–49. University of Amsterdam.

Abzianidze, L. (2015b). A tableau prover for natural logic and language. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2492–2502, Lisbon, Portugal. Association for Computational Linguistics.

Abzianidze, L. (2015c). Towards a wide-coverage tableau method for natural logic. In Murata, T., Mineshima, K., and Bekki, D., editors, *New Frontiers in Artificial Intelligence: JSAI-isAI 2014 Workshops, LENLS, JURISIN, and GABA, Kanagawa, Japan, October 27-28, 2014, Revised Selected Papers*, pages 66–82. Springer Berlin Heidelberg, Berlin, Heidelberg.

Abzianidze, L. (2016). Natural solution to fracas entailment problems. In *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics (*SEM 2016)*, pages 64–74, Berlin, Germany. Association for Computational Linguistics.

Adams, R. (2006). Textual entailment through extended lexical overlap. In *In The Second PASCAL Recognising Textual Entailment Challenge (RTE-2)*.

Ajdukiewicz, K. (1935). Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27. English translation "Syntactic Connexion" by H. Weber in McCall, S. (Ed.) *Polish Logic*, pp. 207–231, Oxford University Press, Oxford, 1967.

Angeli, G. and Manning, C. D. (2014). Naturalli: Natural logic inference for common sense reasoning. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Artzi, Y., Lee, K., and Zettlemoyer, L. (2015). Broad-coverage ccg semantic parsing with amr. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710, Lisbon, Portugal. Association for Computational Linguistics.

Bach, E. and Cooper, R. (1978). The np-s analysis of relative clauses and compositional semantics. *Linguistics and Philosophy*, 2(1):145–150.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The berkeley framenet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98, pages 86–90. Association for Computational Linguistics.

Baldwin, T. and Kim, S. N. (2010). Multiword expressions. In Indurkhya, N. and Damerau, F. J., editors, *Handbook of Natural Language Processing, Second Edition.*, pages 267–292. Chapman and Hall/CRC.

Baldwin, T., Kordoni, V., and Villavicencio, A. (2009). Prepositions in applications: A survey and introduction to the special issue. *Computational Linguistics*, 35(2):119–149.

Bar-Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., and Szpektor, I. (2006). The second pascal recognizing textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognizing Textual Entailment*, pages 1–9.

Bar-Hillel, Y. (1953). A Quasi-Arithmetical Notation for Syntactic Description. *Language*, 29(1):47–58.

Barwise, J. and Cooper, R. (1981). Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4(2):159–219.

Beltagy, I. and Erk, K. (2015). On the proper treatment of quantifiers in probabilistic logic semantics. In *Proceedings of the 11th International Conference on Computational Semantics (IWCS-2015)*, London, UK.

Beltagy, I., Roller, S., Boleda, G., Erk, K., and Mooney, R. (2014). Utexas: Natural language semantics using distributional semantics and probabilistic logic. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 796–801, Dublin, Ireland. Association for Computational Linguistics and Dublin City University.

Beltagy, I., Roller, S., Cheng, P., Erk, K., and Mooney, R. J. (2015). Representing meaning with a combination of logical form and vectors. *CoRR*, abs/1505.06816.

Benthem, J. v. (1991). *Language in action : categories, lambdas and dynamic logic*. Studies in logic and the foundations of mathematics. North-Holland.

Bertot, Y., Huet, G., Castéran, P., and Paulin-Mohring, C. (2010). *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg.

Beth, E. W. (1955). Semantic Entailment and Formal Derivability. *Koninklijke Nederlandse Akademie van Wentenschappen, Proceedings of the Section of Sciences*, 18:309–342.

Bjerva, J., Bos, J., Van der Goot, R., and Nissim, M. (2014). The meaning factory: Formal semantics for recognizing textual entailment and determining semantic similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 642–646, Dublin, Ireland.

Blackburn, P. and Bos, J. (2005). *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI.

Bos, J. (2008). Wide-coverage semantic analysis with boxer. In Bos, J. and Delmonte, R., editors, *Semantics in Text Processing. STEP 2008 Conference Proceedings*, Research in Computational Semantics, pages 277–286. College Publications.

Bos, J. (2009). Towards a large-scale formal semantic lexicon for text processing. In Chiarcos, C., Eckart de Castilho, R., and Stede, M., editors, *From Form to Meaning: Processing Texts Automatically. Proceedings of the Biennal GSCL Conference 2009*, pages 3–14.

Bos, J., Clark, S., Steedman, M., Curran, J. R., and Hockenmaier, J. (2004). Wide-coverage semantic representations from a ccg parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING '04)*, pages 1240–1246, Geneva, Switzerland.

Bos, J. and Markert, K. (2005). 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.

Bos, J. and Markert, K. (2006). When logical inference helps determining textual entailment (and when it doesn't). In Magnini, B. and Dagan, I., editors, *The Second PASCAL Recognising Textual Entailment Challenge. Proceedings of the Challenges Workshop*, pages 98–103, Venice, Italy.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015a). A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Bowman, S. R., Potts, C., and Manning, C. D. (2015b). Recursive neural networks can learn logical semantics. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 12–21, Beijing, China. Association for Computational Linguistics.

Carroll, J., Briscoe, T., and Sanfilippo, A. (1998). Parser evaluation: a survey and a new proposal. In *Proceedings, First International Conference on Language Resources and Evaluation*, pages 447–454. European Language Resources Association.

Champollion, L. (2010). Quantification and negation in event semantics. In Partee, B. H., Glanzberg, M., and Skilters, J., editors, *Formal Semantics and Pragmatics: Discourse, Context, and Models*, volume 6 of *Baltic International Yearbook of Cognition, Logic and Communication*, pages 1–23. New Prairie Press.

Champollion, L. (2014). The interaction of compositional semantics and event semantics. *Linguistics and Philosophy*, pages 1–36.

Chomsky, N. (1986). *Knowledge of Language: Its Nature, Origin, and Use*. Praeger.

Church, A. (1940). A formulation of the simple theory of types. *Jurnal of Symbolic Logic*, 5(2):56–68.

Clark, S. and Curran, J. (2003). Log-linear models for wide-coverage ccg parsing. In Collins, M. and Steedman, M., editors, *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 97–104.

Clark, S. and Curran, J. R. (2004a). The importance of supertagging for wide-coverage ccg parsing. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Clark, S. and Curran, J. R. (2004b). Parsing the wsj using ccg and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*.

Clark, S. and Curran, J. R. (2007). Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, 33.

Clark, S., Steedman, M., and Curran, J. R. (2004). Object-extraction and question-parsing using ccg. In Lin, D. and Wu, D., editors, *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 111–118, Barcelona, Spain. Association for Computational Linguistics.

Constable, J. and Curran, J. (2009). Integrating verb-particle constructions into ccg parsing. In *Proceedings of the Australasian Language Technology Association Workshop 2009*, pages 114–118, Sydney, Australia.

Cooper, R. (1983). *Quantification and Syntactic Theory*, volume 21 of *Synthese Language Library*. Springer Netherlands.

Cooper, R., Crouch, D., Eijck, J. V., Fox, C., Genabith, J. V., Jaspars, J., Kamp, H., Milward, D., Pinkal, M., Poesio, M., Pulman, S., Briscoe, T., Maier, H., and Konrad, K. (1996). *FraCaS: A Framework for Computational Semantics*. Deliverable D16.

Curran, J., Clark, S., and Bos, J. (2007). 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, Prague, Czech Republic. Association for Computational Linguistics.

Curran, J. R. and Clark, S. (2003a). Investigating gis and smoothing for maximum entropy taggers. In *Proceedings of the Tenth Conference on European Chapter of the Association for Computational Linguistics - Volume 1*, EACL '03, pages 91–98, Stroudsburg, PA, USA. Association for Computational Linguistics.

Curran, J. R. and Clark, S. (2003b). Language independent ner using a maximum entropy tagger. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 164–167, Stroudsburg, PA, USA. Association for Computational Linguistics.

Curry, H. B. and Feys, R. (1958). *Combinatory Logic, Volume I*. North-Holland. Second printing 1968.

Dagan, I., Glickman, O., and Magnini, B. (2006). The pascal recognising textual entailment challenge. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*.

Dagan, I., Roth, D., Sammons, M., and Zanzotto, F. M. (2013). *Recognizing Textual Entailment: Models and Applications*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

D'Agostino, M., Gabbay, D. M., Hähnle, R., and Posegga, J., editors (1999). *Handbook of Tableau Methods*. Springer Netherlands, Dordrecht.

Davidson, D. (1967). The logical form of action sentences. In Rescher, N., editor, *The Logic of Decision and Action*, pages 81–120. Univ. of Pittsburgh Press.

de Groote, P. (2001). Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 252–259. Association for Computational Linguistics.

Djordjevic, B., Curran, J., and Clark, S. (2007). *Proceedings of the Tenth International Conference on Parsing Technologies*, chapter Improving the Efficiency of a Wide-Coverage CCG Parser, pages 39–47. Association for Computational Linguistics.

Dong, Y., Tian, R., and Miyao, Y. (2014). Encoding generalized quantifiers in dependency-based compositional semantics. In *Proceedings of the 28th Pacific Asia Conference on Language, Information, and Computation*, pages 585–594, Phuket,Thailand. Department of Linguistics, Chulalongkorn University.

Dowty, D. (1994). The role of negative polarity and concord marking in natural language reasoning. *Semantics and Linguistic Theory*, 4(0).

Eijck, J. V. (2005). Syllogistics = monotonicity + symmetry + existential import.

Erk, K. (2012). Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, 6(10):635–653.

Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*. MIT Press.

Fitting, M. (1990). *First-order Logic and Automated Theorem Proving*. Springer-Verlag New York, Inc., New York, NY, USA.

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

Fyodorov, Y., Winter, Y., and Francez, N. (2003). Order-based inference in natural logic. *Logic Journal of the IGPL*, 11(4):385–416.

Gallin, D. (1975). *Intensional and Higher-Order Modal Logic: With Applications to Montague Semantics*. American Elsevier Pub. Co.

Ganitkevitch, J., VanDurme, B., and Callison-Burch, C. (2013). PPDB: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2013)*, Atlanta, Georgia. Association for Computational Linguistics.

Giampiccolo, D., Dang, H. T., Magnini, B., Dagan, I., Cabrio, E., and Dolan, B. (2008). The fourth PASCAL recognizing textual entailment challenge. In *Proceedings of the First Text Analysis Conference, TAC 2008, Gaithersburg, Maryland, USA, November 17-19, 2008*.

Goré, R. (1999). Tableau methods for modal and temporal logics. In D'Agostino, M., Gabbay, D. M., Hähnle, R., and Posegga, J., editors, *Handbook of Tableau Methods*, pages 297–396. Springer Netherlands, Dordrecht.

Grice, H. (1975). Logic and conversation. In *Syntax and Semantics*. Academic Press.

Groote, P. and Winter, Y. (2015). A type-logical account of quantification in event semantics. In Murata, T., Mineshima, K., and Bekki, D., editors, *New Frontiers in Artificial Intelligence: JSAI-isAI 2014 Workshops, LENLS, JURISIN, and GABA, Kanagawa, Japan, October 27-28, 2014, Revised Selected Papers*, pages 53–65. Springer Berlin Heidelberg, Berlin, Heidelberg.

Harabagiu, S. and Hickl, A. (2006). Methods for using textual entailment in open-domain question answering. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 905–912. Association for Computational Linguistics.

Harris, Z. (1955). Distributional structure. *Word*, 10(23):146–162.

Heim, I. and Kratzer, A. (1998). *Semantics in generative grammar*. Blackwell textbooks in linguistics. Blackwell publishers, Cambridge (Mass.), Oxford.

Hemann, J., Swords, C., and Moss, L. (2015). Two advances in the implementations of extended syllogistic logics. In Balduccini1, M., Mileo, A., Ovchinnikova, E., Russo, A., and Schüuller, P., editors, *Joint Proceedings of the 2nd Workshop on Natural Language Processing and Automated Reasoning, and the 2nd International Workshop on Learning and Nonmonotonic Reasoning at LPNMR 2015*, pages 1–14.

Henkin, L. (1950). Completeness in the theory of types. *J. Symbolic Logic*, 15(2):81–91.

Hintikka, J. (1955). *Two Papers on Symbolic Logic: Form and Content in Quantification Theory and Reductions in the Theory of Types*. Number 8 in Acta philosophica Fennica. Societas Philosophica.

Hobbs, J. R., Stickel, M. E., Appelt, D. E., and Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence*, 63(1-2):69–142.

Hockenmaier, J. (2003). *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh.

Hockenmaier, J. and Steedman, M. (2007). Ccgbank: A corpus of ccg derivations and dependency structures extracted from the penn treebank. *Comput. Linguist.*, 33(3):355–396.

Hoeksema, J. (1983). Negative Polarity and the Comparative. *Natural Language and Linguistic Theory*, 1(3):403–434.

Honnibal, M., Curran, J. R., and Bos, J. (2010). Rebanking ccgbank for improved np interpretation. In *Proceedings of the 48th Meeting of the Association for Computational Linguistics (ACL 2010)*, pages 207–215, Uppsala, Sweden.

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

Icard, T. F. and Moss, L. S. (2014). Recent progress on monotonicity. *Linguistic Issues in Language Technology*, 9.

Jimenez, S., Dueñas, G., Baquero, J., and Gelbukh, A. (2014). Unal-nlp: Combining soft cardinality features for semantic textual similarity, relatedness and entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 732–742, Dublin, Ireland. Association for Computational Linguistics and Dublin City University.

Kamp, H. and Partee, B. (1995). Prototype theory and compositionality. *Cognition*, 57(2):129–191.

Kamp, H. and Reyle, U. (1993). *From discourse to logic; an introduction to modeltheoretic semantics of natural language, formal logic and DRT*. Dordrecht: Kluwer.

Karttunen, L. (1971). Implicative Verbs. *Language*, 47(2):340–358.

Karttunen, L. (2012). 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, Montréal, Canada. Association for Computational Linguistics.

Karttunen, L. (2015). From natural logic to natural reasoning. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing: 16th International Conference, CICLing 2015, Cairo, Egypt, April 14-20, 2015, Proceedings, Part I*, pages 295–309. Springer International Publishing.

Keller, W. R. (1988). Nested cooper storage: The proper treatment of quantification in ordinary noun phrases. In Reyle, U. and Rohrer, C., editors, *Natural Language Parsing and Linguistic Theories*, pages 432–447. Springer Netherlands, Dordrecht.

Kotlerman, L., Dagan, I., Szpektor, I., and Maayan, Z.-G. (2010). Directional distributional similarity for lexical inference. *Natural Language Engineering*, 16:359–389.

Kruszewski, G. and Baroni, M. (2015). So similar and yet incompatible: Toward the automated identification of semantically compatible words. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 964–969, Denver, Colorado. Association for Computational Linguistics.

Kubler, S., McDonald, R., Nivre, J., and Hirst, G. (2009). *Dependency Parsing*. Morgan and Claypool Publishers.

Lacatusu, F., Hickl, A., Roberts, K., Shi, Y., Bensley, J., Rink, B., Wang, P., and Taylor, L. (2006). Lcc's gistexter at duc 2006: Multi-strategy multi-document summarization. In *in Proceedings of DUC 2006*.

Lai, A. and Hockenmaier, J. (2014). Illinois-lh: A denotational and distributional approach to semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 329–334, Dublin, Ireland. Association for Computational Linguistics and Dublin City University.

Lakoff, G. (1970). Linguistics and natural logic. In Davidson, D. and Harman, G., editors, *Semantics of Natural Language*, volume 40 of *Synthese Library*, pages 545–665. Springer Netherlands.

Lambek, J. (1958). The mathematics of sentence structure. *Americal Mathematical Monthly*, 65:154–170.

Lapata, M. and Keller, F. (2005). Web-based models for natural language processing. *ACM Transactions on Speech and Language Processing*, 2(1).

Lenat, D. B. and Guha, R. V. (1989). *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Levy, O., Remus, S., Biemann, C., and Dagan, I. (2015). Do supervised distributional methods really learn lexical inference relations? In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 970–976. Association for Computational Linguistics.

Lewis, M. and Steedman, M. (2013). Combined distributional and logical semantics. *Transactions of the Association for Computational Linguistics (TACL)*, 1:179–192.

Lewis, M. and Steedman, M. (2014a). 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. Association for Computational Linguistics.

Lewis, M. and Steedman, M. (2014b). Improved CCG parsing with semi-supervised supertagging. *Transactions of the Association for Computational Linguistics (TACL)*, 2:327–338.

Liang, P., Jordan, M. I., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 590–599.

Lis, Z. (1960). Wynikanie semantyczne a wynikanie formalne (logical consequence, semantic and formal). *Studia Logica*, 10(1):39–60.

MacCartney, B. (2009). *Natural language inference*. Phd thesis, Stanford University.

MacCartney, B. and Manning, C. D. (2007). Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, RTE '07, pages 193–200, Stroudsburg, PA, USA. Association for Computational Linguistics.

MacCartney, B. and Manning, C. D. (2008). Modeling semantic containment and exclusion in natural language inference. In Scott, D. and Uszkoreit, H., editors, *COLING*, pages 521–528.

MacCartney, B. and Manning, C. D. (2009). An extended model of natural logic. In *Proceedings of the Eighth International Conference on Computational Semantics*, IWCS-8 '09, pages 140–156. Association for Computational Linguistics.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60. Association for Computational Linguistics.

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

Marelli, M., Menini, S., Baroni, M., Bentivogli, L., Bernardi, R., and Zamparelli, R. (2014a). Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of SemEval 2014 (International Workshop on Semantic Evaluation)*, pages 1–8, East Stroudsburg PA. ACL.

Marelli, M., Menini, S., Baroni, M., Bentivogli, L., Bernardi, R., and Zamparelli, R. (2014b). A sick cure for the evaluation of compositional distributional semantic models. In Calzolari, N., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA).

Martins, A., Almeida, M., and Smith, A. N. (2013). Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622. Association for Computational Linguistics.

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

McDonald, R., Pereira, F., Ribarov, K., and Hajic, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*.

Mehdad, Y., Negri, M., and Federico, M. (2010). Towards cross-lingual textual entailment. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 321–324. Association for Computational Linguistics.

Merlo, P. and Ferrer, E. E. (2006). The notion of argument in prepositional phrase attachment. *Computational Linguistics*, 32(3):341–378.

Miller, G. A. (1995). Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41.

Miller, G. A., Leacock, C., Tengi, R., and Bunker, R. T. (1993). A semantic concordance. In *Proceedings of the Workshop on Human Language Technology*, HLT '93, pages 303–308. Association for Computational Linguistics.

Mineshima, K., Martínez-Gómez, P., Miyao, Y., and Bekki, D. (2015). Higher-order logical inference with compositional semantics. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2055–2061, Lisbon, Portugal. Association for Computational Linguistics.

Minnen, G., Carroll, J., and Pearce, D. (2001). Applied morphological processing of english. *Nat. Lang. Eng.*, 7(3):207–223.

Montague, R. (1970). Universal grammar. *Theoria*, 36(3):373–398.

Montague, R. (1973). The proper treatment of quantification in ordinary English. In Hintikka, K. J. J., Moravcsic, J., and Suppes, P., editors, *Approaches to Natural Language*, pages 221–242. Reidel, Dordrecht.

Moss, L. S. (2010a). Logics for natural language inference. Expanded version of lecture notes from a course at ESSLLI 2010.

Moss, L. S. (2010b). Natural logic and semantics. In Aloni, M., Bastiaanse, H., de Jager, T., and Schulz, K., editors, *Logic, Language and Meaning: 17th Amsterdam Colloquium, Amsterdam, The Netherlands, December 16-18, 2009, Revised Selected Papers*, pages 84–93. Springer Berlin Heidelberg, Berlin, Heidelberg.

Moss, L. S. (2011). Syllogistic logic with comparative adjectives. *Journal of Logic, Language and Information*, 20(3):397–417.

Moss, L. S. (2012). The soundness of internalized polarity marking. *Studia Logica*, 100(4):683–704.

Muskens, R. (1996). Combining montague semantics and discourse representation. *Linguistics and Philosophy*, 19(2):143–186.

Muskens, R. (2001). Categorial grammar and lexical-functional grammar. In Butt, M. and King, T. H., editors, *Proceedings of the LFG01 Conference*, pages 259–279. CSLI Publications.

Muskens, R. (2003). Language, Lambdas, and Logic. In Kruijff, G.-J. and Oehrle, R., editors, *Resource Sensitivity in Binding and Anaphora*, Studies in Linguistics and Philosophy, pages 23–54. Kluwer.

Muskens, R. (2010). An analytic tableau system for natural logic. In Aloni, M., Bastiaanse, H., de Jager, T., and Schulz, K., editors, *Logic, Language and Meaning*, volume 6042 of *Lecture Notes in Computer Science*, pages 104–113. Springer Berlin Heidelberg.

Muskens, R. (2011). Towards logics that model natural reasoning. Program Description.

Muskens, R. A. (1995). *Meaning and Partiality*. CSLI Publications, Stanford.

Nairn, R., Condoravdi, C., and Karttunen, L. (2006). Computing relative polarity for textual inference. In *Proceedings of the Fifth International Workshop on Inference in Computational Semantics (ICoS-5)*.

Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys*, 41(2):1–69.

Nivre, J. (2010). Dependency parsing. *Language and Linguistics Compass*, 4(3):138–152.

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., and Zeman, D. (2016). Universal dependencies v1: A multilingual treebank collection. In Chair), N. C. C., Choukri, K., Declerck, T., Goggi, S., Grobelnik, M., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Association (ELRA).

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13:95–135.

Pado, S., Galley, M., Jurafsky, D., and Manning, D. C. (2009). Robust machine translation evaluation with entailment features. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 297–305. Association for Computational Linguistics.

Palmer, M., Kingsbury, P., and Gildea, D. (2005). The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31.

Parsons, T. (1972). Some problems concerning the logic of grammatical modifiers. In Davidson, D. and Harman, G., editors, *Semantics of Natural Language*, pages 127–141. Springer Netherlands, Dordrecht.

Parsons, T. (1990). *Events in the semantics of English : a study in subatomic semantics / Terence Parsons*. MIT Press Cambridge, Mass.

Partee, B. (1975). Montague Grammar and Transformational Grammar. *Linguistic Inquiry*, 6(2):203–300.

Partee, B. H. (1987). Noun phrase interpretation and type-shifting principles. *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers*, 8:115–143.

Partee, B. H. (2001). Privative adjectives: subsective plus coercion. In Bauerle, R., Reyle, U., and Zimmermann, T., editors, *Presuppositions and Discourse: Essays Offered to Hans Kamp*, volume 21 of *Current Research in the Semantics / Pragmatics Interface*, pages 273–285. BRILL.

Potts, C. (2015). Presupposition and implicature. In Lappin, S. and Fox, C., editors, *The Handbook of Contemporary Semantic Theory*, pages 168–202. Wiley-Blackwell, 2 edition.

Proisl, T., Evert, S., Greiner, P., and Kabashi, B. (2014). Semantiklue: Robust semantic similarity at multiple levels using maximum weight matching. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 532–540, Dublin, Ireland. Association for Computational Linguistics and Dublin City University.

Raina, R., Ng, A. Y., and Manning, C. D. (2005). Robust textual inference via learning and abductive reasoning. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 1099–1105.

Ratnaparkhi, A., Reynar, J., and Roukos, S. (1994). A maximum entropy model for prepositional phrase attachment. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 250–255. Association for Computational Linguistics.

Russell, B. (1905). On denoting. *Mind*, 14(56):479–493.

Ruys, E. and Winter, Y. (2011). Quantifier scope in formal linguistics. In Gabbay, M. D. and Guenthner, F., editors, *Handbook of Philosophical Logic: Volume 16*, pages 159–225. Springer Netherlands, Dordrecht.

Sag, I. A., Baldwin, T., Bond, F., Copestake, A., and Flickinger, D. (2001). Multiword expressions: A pain in the neck for nlp. In *In Proc. of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2002*, pages 1–15.

Sánchez-Valencia, V. (1991). Categorial grammar and natural reasoning. ILTI Publication Series for Logic, Semantics, and Philosophy of Language LP-91-08, University of Amsterdam.

Schuler, K. K. (2005). *Verbnet: A Broad-coverage, Comprehensive Verb Lexicon*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA. AAI3179808.

Shwartz, V., Goldberg, Y., and Dagan, I. (2016). Improving hypernymy detection with an integrated path-based and distributional method. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2389–2398. Association for Computational Linguistics.

Smullyan, R. M. (1968). *First-order Logic*. Springer-Verlag.

Steedman, M. (1996). *Surface structure and interpretation*. Linguistic inquiry monographs, 30. MIT Press.

Steedman, M. (2000). *The Syntactic Process*. MIT Press, Cambridge, MA, USA.

Steedman, M. and Baldridge, J. (2011). Combinatory Categorial Grammar. In Borsley, Robert, D. and Börjars, K., editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, pages 181–224. Wiley-Blackwell.

Strawson, P. F. (1950). On referring. *Mind*, 59(235):320–344.

Tesnière, L. (1959). *Elements de syntaxe structurale*. Editions Klincksieck.

Tian, R., Miyao, Y., and Matsuzaki, T. (2014). Logical inference on dependency-based compositional semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 79–89, Baltimore, Maryland. Association for Computational Linguistics.

Turian, J., Ratinov, L.-A., and Bengio, Y. (2010). Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. Association for Computational Linguistics.

Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188.

van Benthem, J. (1986). *Essays in Logical Semantics*, volume 29 of *Studies in Linguistics and Philosophy*. Springer Netherlands.

van Benthem, J. (1987). Meaning: Interpretation and inference. *Synthese*, 73(3):451–470.

van Benthem, J. (2008a). A brief history of natural logic. In *Technical Report PP-2008-05*. Institute for Logic, Language & Computation.

van Benthem, J. (2008b). Natural logic: A view from the 1980s. In M. K. Chakraborty, B. Lowe, M. N. Mitra and S. Sarukkai, editor, *Logic, Navya-Nayaya & Applications. Homage to Bimal Krishna Matilal*, volume 15 of *Studies in Logic*. London College Publications.

van Eijck, J. (2007). Natural logic for natural language. In ten Cate, B. D. and Zeevat, H. W., editors, *Logic, Language, and Computation: 6th International Tbilisi Symposium on Logic, Language, and Computation,TbiLLC 2005 Batumi, Georgia, September 12-16, 2005. Revised Selected Papers*, pages 216–230. Springer Berlin Heidelberg, Berlin, Heidelberg.

von Fintel, K. (1999). NPI licensing, strawson entailment, and context dependency. *Journal of Semantics*, 16:99–148.

Winter, Y. and Zwarts, J. (2011). Event semantics and abstract categorial grammar. In Kanazawa, M., Kornai, A., Kracht, M., and Seki, H., editors, *The Mathematics of Language*, volume 6878 of *Lecture Notes in Computer Science*, pages 174–191. Springer Berlin Heidelberg.

Yin, W., Schütze, H., Xiang, B., and Zhou, B. (2015). ABCNN: attention-based convolutional neural network for modeling sentence pairs. *CoRR*, abs/1512.05193.

Zamansky, A., Francez, N., and Winter, Y. (2006). A 'natural logic' inference system using the lambek calculus. *Journal of Logic, Language and Information*, 15(3):273–295.

Zhao, J., Zhu, T., and Lan, M. (2014). Ecnu: One stone two birds: Ensemble of heterogenous measures for semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 271–277, Dublin, Ireland. Association for Computational Linguistics and Dublin City University.

Zwarts, F. (1981). Negatief polaire uitdrukkingen i. *GLOT*, 4:35–132.