Formal Approaches to Textual Entailment, Polarity, and Pronominal Anaphora

By:

M. C. José de Jesús Lavalle Martínez

A dissertation submitted in partial fulfillment
of the requirements for the degree of:

DOCTOR OF SCIENCE IN COMPUTER SCIENCE

at

Instituto Nacional de Astrofísica, Óptica y Electrónica

October, 2017
Tonantzintla, Puebla

Supervised by:

Dr. Manuel Montes y Gómez, INAOE
Dr. Héctor Jiménez Salazar, UAM Cuajimalpa

Contents

List of Figures

List of Tables

Acknowledgments

I want to express my gratitude to Manuel Montes y Gómez, Héctor Jiménez Salazar and Luis Villaseñor Pineda for the long discussions, support and guidance for four years.

I also want to express my gratitude to Jesús Ariel Carrasco Ochoa, Aurelio López López and Gustavo Rodríguez Gómez, members of my tutorial committee, their comments have been very helpful for the last three years.

I also want to thank Jesús Antonio González Bernal and Felipe Orihuela Espina; I enjoyed very much their courses about Machine Learning and Research Seminar, respectively.

Manuel Sabino Lazo Cortés and José Francisco Martínez Trinidad have been very supportive in many ways.

I also want to thank Mauricio Javier Osorio Galindo, for all the discussions about logic that we have kept for a very long time.

Last but not least, Emma Isabel Altamirano López, Bernardo Lavalle Martínez and Alfredo Lavalle Martínez have played an invaluable role in this work.

I have to mention that this research was partly supported by PRODEP-SEP under Grant PROMEP/103.5/13/5618, by Benemérita Universidad Autónoma de Puebla under Grant BUAP-803, and by CONACYT under the Thematic Networks Program (Language Technologies Thematic Network Projects 260178 and 271622).

This work is dedicated to my beloved and lovely daughters, “Las Isas”.

Abstract

This dissertation is about the interaction of computational semantics and formal methods, in the aim of finding answers to the question: How does natural language work? On the last three decades, statistical and machine learning methods have taken command giving good results in solving very specific tasks. Unfortunately, good results does not mean good understanding. To know that a ninety per cent of the times I will get certain result does not mean that I am comprehending the phenomenon under study.

This work shows how three aspects of computational semantics have been solved from the point of view of formal methods. The main goal was to solve the problem of Recognizing Textual Entailment, but very soon aspects of polarity computing and anaphora resolution had to be considered as well. Fortunately, a characterization of the entailments that can be done with Natural Logic have been found. Also, the computable concept of polarity has coincided with respect to the algorithms analyzed. About pronominal anaphora resolution it should be said that a method to solve anaphora in parsing time has been found, without the need of a logical form. The method can solve Donkey, intra and intersentencial anaphora.

Resumen

Esta tesis trata de la interacción entre semántica computacional y métodos formales, con el objetivo de encontrar respuestas a la pregunta: ¿Cómo funciona el lenguaje natural? En las últimas tres décadas, los métodos estadísticos y de aprendizaje computacional han dominado y dado buenos resultados al resolver tareas muy específicas. Desafortunadamente, buenos resultados no significan buen entendimiento. Saber que el noventa por ciento de las veces se obtiene cierto resultado no significa que se está comprendiendo el fenómeno estudiado.

Este trabajo muestra cómo tres aspectos de la semántica computacional han sido resueltos desde el punto de vista de los métodos formales. El principal objetivo era resolver el problema del Reconocimiento de Implicación Textual, pero muy pronto también se tuvieron que considerar aspectos como el cálculo de polaridad y la resolución de anáfora. Afortunadamente, se ha encontrado una caracterización de las implicaciones que se pueden hacer con Lógica Natural. También, que el concepto computable de polaridad coincide con respecto a los algoritmos analizados. Sobre la resolución de anáfora pronominal se debe decir que se ha encontrado un método para resolverla al momento de hacer el análisis sintáctico, sin necesitar alguna forma lógica. El método puede resolver anáforas tipo *Donkey*, intra e intersentenciales.

Acronyms

RTE Recognizing Textual Entailment . . . 11

COGEX a logic prover for question answering . . . 16

OTTER an automated theorem prover for first order and equational logic . . . 17

FOL First Order Logic . . . 19

NL Natural Logic . . . 19

DRT Discourse Representation Theory . . . 25

DPL Dynamic Predicate Logic . . . 26

ML a strong typed functional programming language called Meta Language . . . 63

POS Part Of Speech . . . 102

Chapter 1

Introduction

Textual entailment is defined [DGM06] as a directional relationship between pairs of text expressions, denoted by t the entailing "Text", and h the entailed "Hypothesis". We say that t entails h if the meaning of h can be inferred from the meaning of t, as would typically be interpreted by people.

Recognizing Textual Entailment (RTE) is essential for other Natural Language Processing tasks such as: Semantic Search, Question Answering, Text Summarization, and Information Extraction. Even more, there is the expectation that in getting better methods of textual entailment, these will improve our understanding of how a machine can assign meaning to natural language.

Different methods have been used to solve the RTE problem [DRSZ13], those methods are based on machine learning, linear programming, probabilistic calculus, optimization, and logic.

Without a doubt, the statistical approach has dominated in the solution of Natural Language Processing tasks in the last two decades. Nevertheless, this dissertation is about formal semantics applied to problems in computational linguistics, where the explanatory power is required to understand the linguistic phenomena.

After the bibliographic revision, and focusing on the logical methods to solve RTE, it was found that all the methods ended up deciding through a machine learning algorithm, so the explanatory power of the logical formalisms was lost. But, one of those methods called our attention, the one of MacCartney and Manning [MM07] entitled *Natural Logic for Textual Inference*. The following four aspects are remarkable.

First, Natural Logic[1] does not produce logical forms. The main objection to use logical formalisms have been that the coding of natural language sentences in some logical form overcome practical expectations. The same objection existed in the industry of software

[1]Natural Logic uses the syntactic structure of a sentence and the monotonic properties of its functors in order to compute the composition (called polarity) of those functors, if a subexpression of the sentence has positive/negative polarity it can be changed for a greater/lesser subexpression without changing the truth value of the resulting sentence.

in the 1990s, but it has faded away with the advent of model checking[2].

Second, Natural Logic was created without a model-theoretic semantics. Hence, in order to recover its explanatory power, we define a model-theoretic semantics for Natural Logic, and from there a model checking method was built.

Third, the authors reported that they were not able to compute polarity because they had to use the Stanford parser. After a bibliographic review, it was found that there are four main algorithms to compute polarity, the ones of van Benthem [vB86, vB91], Sánchez [SV91], Dowty [Dow94], and van Eijck [vE07].

Sánchez based its algorithm on the algorithm of van Benthem, Sánchez proved that his algorithm was sound with respect to certain semantics, but he did not prove its relationship with the algorithm of van Benthem. Dowty said that his algorithm was more appropiate, than the algorithm of Sánchez, for some linguistic investigations, but he did not prove it. van Eijck criticised the previous algorithms but never proved why and in what sense his algorithm was a better one. For all these facts, we investigate the relationship that the four algorithms have, proving that they are equivalent.

Fourth, the method of MacCartney and Manning is not able to solve anaphora and, according to Sammons [Sam11], the linguistic phenomenum of anaphora is the one that appears most in every day and in RTE, even worse it is the main source of error in RTE systems. In solving anaphora, methods based on probability, clustering and machine learning techniques [Mit02] have been created. But, also logical formalisms have been created to solve anaphora, these methods are inspired by Dynamic Propositional Logic [HTK00], a logic that was developed to reason about the change of state in imperative programs. We also took ideas from Dynamic Propositional Logic to dynamize an extension of *AB* grammars and solve anaphora.

Chapter 2 introduces the reader to the essential concepts which permit to understand the main contributions in this dissertation, also what other people have formally done in recognizing textual entailment, computing polarity and solving anaphora is presented.

In Chapter 3 the linguistic phenomenon of polarity is introduced, the algorithms to compute it are presented and explained. Next, their equivalence is proved.

In Chapter 4 a model-theoretic semantics for Natural Logic is developed. Its soundness and completeness with respect to the proof theory of *AB* grammars[3] and Natural Logic

[2]Intuitively, model checking is a method based on formal semantics that decides whether a property is met. But, its main purpose is to find out why a property is not met.

[3]*AB* grammars are built from a set of primitive types (in our case e and t), and functors are constructed using the operators / and \, these operators are intended to distinguish if the argument of a functor is either on the right or on the left, respectively. In order to decide whether a natural language expression is a

are demonstrated. An automatic theorem prover for an extension of *AB* grammars is constructed, and a model checking method for Natural Logic is developed. With these tools is recovered the explanatory power of Natural Logic in RTE.

Chapter 5 contains our proposal to solve pronominal anaphora. Conditional sentences and relative clauses are added to *AB* grammars, in such a way that our proposal can resolve Donkey[4], intra and intersentencial anaphora. A method based on model checking is developed to resolve pronominal anaphora.

Closure contains the conclusions about the achievements reached, a future work agenda, and the tangible contributions of the research.

sentence, the type t has to be derived by modus ponens from the types of each word in the sentence.

[4]Donkey anaphora is a relation where a pronoun cannot refer to an indefinite noun phrase, because the pronoun, after the sentence is passed to a logical form, is not in the scope of the indefinite noun phrase which is bound by a quantifier. The most famous instance of Donkey anaphora is: *Every farmer who owns a donkey beats it*.

Chapter 2

Background

This chapter introduces the reader to the essential concepts which permit to understand the main contributions in this dissertation, also what other people have formally done in recognizing textual entailment, computing polarity and solving anaphora is presented.

First, Textual Entailment is defined and some approaches to recognize it are mentioned. Besides, our view about textual entailment is stated, i.e., it is important to know if a text entails a hypothesis, but it is more important to know why it does not. Also, five methods that use some kind of logic to recognize textual entailment are briefly discussed.

Second, it can be said that Natural Logic is any kind of Categorial Grammar with modus ponens as the unique inference rule plus reasoning through polarity, in this case the categorial grammar presented is the calculus of Ajdukiewicz [Ajd78, Bac88, MR12]. Also, the static concept of monotonicity of types is defined, in order to set the dynamic concept of polarity. As well as, the aspects that were the motivation to find out the relation between the polarity algorithms, discussed in chapter 3, are mentioned.

The chapter ends with the definition of the problem of anaphora resolution, the knowledge required to solve it, and some approaches of solution are mentioned. Two logical methods are emphasized, namely Discourse Representation Theory and Dynamic Predicate Logic, also some citations to ways of dynamise higher order logics are done.

2.1 Recognizing Textual Entailment

Textual entailment is defined [DGM06] as a directional relationship between pairs of text expressions, denoted by t the entailing "Text", and h the entailed "Hypothesis". We say that t entails h if the meaning of h can be inferred from the meaning of t, as would typically be interpreted by people.

For example, if t = *At the end of the year, all solid companies pay dividends*, and h = *At the end of the year, all solid insurance companies pay dividends*, then it can be said that t entails h. As an example where t does not entail h, it can be considered the

sentences t = John owns a car, and h = John bought a Jeep.

RTE is essential for other Natural Language Processing tasks such as: Semantic Search, Question Answering, Text Summarization, and Information Extraction. Different methods have been used to solve the RTE problem [DRSZ13], those methods are based on machine learning, linear programming, probabilistic calculus, optimization, and logic.

One could expect that logical methods have the advantage of answering why the entailment was not carried out, but even logical methods decide on the entailment using machine learning, or some kind of optimization. For example, MacCartney and Manning have a system based on Natural Logic to recognize textual entailment [MM07], but their system decides on it by a decision tree. Therefore, the scope of Natural Logic is overriden by the machine learning algorithm.

Natural Logic was developed to reason in natural language without having to use formal systems as predicate calculus or high order logic [vB07, SV91]. Natural Logic only uses lexical, syntactic, and the basic semantic information of a language.

Natural Logic was created on purpose without a Model Theory. However, we are interested in characterizing the scope of Natural Logic in Recognizing Textual Entailment, for this reason we define a Semantics for Natural Logic in the style of Model Theory, and a method of model Checking for Natural Logic.

With respect to model theories of Logics for Natural Language, Moss [Mos10] has done important contributions about fragments of language. Also, Moss [Mos12] has developed a theory of models for polarity reasoning based on preorders and their opposites, mainly to make the polarity algorithm of Dowty sound, but he does not define a Model Checking method for Natural Logic.

The methods discussed in this section use some kind of inference mechanism to recognize textual entailment, except for the one of MacCartney and Manning, which is included because it is based on Natural Logic.

2.1.1 Using COGEX

The system of Hodges et al. [HCFM06] transforms the input text and hypothesis into logical forms. The transformation process includes part-of-speech tagging, parse tree generation, word sense disambiguation and semantic relations detection.

In order to use a logic prover for question answering (COGEX), a list of clauses called "set of support" is required, this is used to begin the search for inferences. Another list, called the usable list, contains clauses used by COGEX to produce inferences. The axioms are about knowledge of the world, linguistic rewriting rules, and synsets of WordNet.

The clauses in the set of support are weighted, a clause with a lesser weight is prefered to participate in the search. The negated hypothesis (COGEX proves by refutation) is added to the set of support with the largest weight, this guarantees that the hypothesis will be the last clause used in the search.

If a refutation is found the prover ends, if there is not a refutation the predicate arguments are relaxed. If despite arguments relaxation a refutation is not found, predicates are dropped from the negated hypothesis until a refutation is found.

When a refutation is found, a score for it is computed, beginning with a perfect score and subtracting points for axioms used, arguments relaxed, and predicates dropped.

If the score for a refutation is greater than a threshold, then it is considered that the entailment is true, otherwise it is considered false.

2.1.2 Using OTTER

In the proposal of Akhmatova [Akho6], it is used an automated theorem prover for first order and equational logic (OTTER). The meaning of a sentence is represented by the set of atomic propositions contained in it, then the sentences are compared by means of their associated propositions.

A syntax-driven semantic analysis is used to get the atomic propositions associated with a sentence. The output of the parser is used as input for the semantic analyzer; from the output of the analyzer, the representation of the sentence in first order logic, which is called *the logic formula*, can be derived.

For Akhmatova, there are many ways to describe meaning through logical form, but they are rigid and hard to produce. Because of that, a simplified representation is proposed.

The simplified representation is build from: three types of objects $\mathtt{Subj}(x)$, $\mathtt{Obj}(x)$ and $\mathtt{Pred}(x)$, a meaning attaching element $\mathtt{iq}(x, <\mathtt{meaning\ of}\ x>)$, and two variants of relationships $\mathtt{attr}(x,y)$ and $\mathtt{prep}(x,y)$.

Later, usign WordNet, a relatedness score between words is computed from the paths between the senses of the words, the longer the path, the lesser the relatedness is. This score together with knowledge rules are given to the automatic theorem prover OTTER.

If for every proposition in the hypothesis sentence p_{h_i} there is one proposition in the text sentence p_{t_j}, such that $p_{t_j} \rightarrow p_{h_i}$, then the entailment holds, otherwise the entailment does not hold.

2.1.3 Using Abduction

Raina et al. [RNM05] begin constructing a syntactic dependency graph using a parser, hand written rules are used to find the heads of all nodes in the parse tree. The relations represented in the dependency graph are translated into a logical formula representation. Each node in the graph is converted into a logical term which is assigned a constant.

Later, abductive theorem proving is realized by the resolution method, where each abductive assumption, and its degree of plausabilty is quantified as a nonnegative cost using the *assumption cost model*. The objective is to find the proof of minimun cost, which is chosen automatically by a machine learning algorithm.

2.1.4 Using Vampire and Paradox

The approach of Bos and Markert [BM06a, BM06b] is based on what they call *shallow semantic analysis* and *deep semantic analysis*.

Four features are obtained from the shallow semantic analysis, the overlap between words in text and hypothesis, the length of text, the length of hypothesis, and the relative length of hypothesis with respect to the text.

To achieve the deep semantic analysis, they use a robust wide-coverage parser, which produces proof trees of Combinatory Categorial Grammar [SB11]. Afterwards, the proof trees are used to build discourse representation structures, these are the semantic representations from Discourse Representation Theory. Later, the semantic representations are translated into first order logic expressions.

The model checker Paradox and the automatic theorem prover Vampire are used to prove whether or not the text implies the hypothesis. Bos and Markert take two features from the automatic theorem prover, and six from the model checker.

A decision tree is trained with the twelve features, and it is used to decide if the text implies the hypothesis.

2.1.5 Using Natural Logic

MacCartney and Manning [MM07, Mac09] use Natural Logic to avoid logical forms, their system is called *NatLog*. They begin with a linguistic pre-processing, the text and the hypothesis are parsed with the Stanford parser, the main purpose of this step is monotonicity marking; nevertheless, they do not use polarity (see section 2.2) as an inference mechanism.

First Author	Inference Mechanism	Logic	BK	Challenge	Decided by
Hodges	COGEX	FOL	WordNet	RTE-2	Optimization
Akhmatova	OTTER	FOL	WordNet	RTE-1	Unclear
Raina	Abduction	FOL		RTE-1	Machine Learning
Bos	Vampire & Paradox	FOL	WordNet	RTE-1	Machine Learning
MacCartney			WordNet	RTE-3	Machine Learning

Table 2.1: Summary of the main characteristics of some logical approaches to recognize textual entailment.

The second step consists of an alignment between the text and the hypothesis, alignments are represented by sequences of atomic edits over words.

Finally, taking as features the monotonicity information and the sequences of edits, a decision tree is trained.

2.1.6 Brief Analysis of the Methods

As it can be seen in Table 2.1, the methods based on some inference mechanism use First Order Logic (FOL) as a form to represent the text and the hypothesis.

For us, it is unclear the decision mechanism that the system of Akhmatova follows, a relatedness score is computed, but its role in the decision process is never mentioned.

The other methods use a decision process different from the inference mechanism, as it has been explained, hiding why the entailment was not carried out.

2.2 Natural Logic

Sánchez in his Ph. D. dissertation [SV91] formalizes the ideas of van Benthem about Natural Logic (NL) and monotonic reasoning [vB86, vB91]. Even though the origins of NL go back to Aristotle [vB07, Kar15]; the central idea, in the program of NL of van Benthem et al., is that natural language, besides communicating ideas, serves to reason without having to use formal systems as the predicate calculus or the high order logic. The idea is to use the syntactic structure of a sentence, semantic properties of their lexical constituents, and functor constructors over these.

They use the calculus of Ajdukiewicz [Ajd78, Bac88, MR12] for the syntactic analysis

of sentences, this has the basic types e (entities, the type of proper nouns), and t (truth values, the type of sentences), then functors between types are defined to construct more complex types. Thereby, if α and β are types, then (α, β) is the type of the functors with domain α and range β (it is read, functor from α to β). To construct a proof tree (also called derivation tree), the calculus of Ajdukiewicz has only the following inference rule

$$\frac{(\alpha, \beta) \qquad \alpha}{\beta}$$

from the functor of type (α, β) and the argument of type α (either to the right, or to the left of the functor), the type β is derived as the result.

It is assumed that each word in the lexicon has a type, for example: common nouns have type (e, t), transitive verbs have type $(e, (e, t))$, intransitive verbs have type (e, t), adjectives and adverbs have type $((e, t), (e, t))$, noun phrases have type $((e, t), t)$, and determiners have type $((e, t), ((e, t), t))$.

Hence to know whether a sentence is well formed, the last inference rule is used to build a proof tree, if its root is t, then the sentence is well formed, otherwise the sentence is ill-formed.

Nevertheless, as there are words that play different roles (for example, *white* could either be an adjective, a noun, or a verb), if a sentence contains words of this kind, the algorithm that constructs the proof tree for such a sentence would have to try out the different types of each word until the type t has been derived.

The first semantic element of NL is that each type denotes a set, hence D_e denotes the set of entities, D_t denotes the set $\{0, 1\}$, $D_{(\alpha,\beta)}$ denotes the set whose elements are functions from α to β, and the following partial order relation is defined on each type.

Definition 2.2.1. [Dow94] Partial order relations on the denotations of types can be defined in the following way:

1. If $d, d' \in D_e$, then $d \leq_e d'$ if and only if $d = d'$,
2. If $d, d' \in D_t$, then $d \leq_t d'$ if and only if $d = 0$ or $d' = 1$,
3. If $d, d' \in D_{(\alpha,\beta)}$, then $d \leq_{(\alpha,\beta)} d'$ if and only if for all $x \in D_\alpha$, $d(x) \leq_\beta d'(x)$. □

The second semantic element of NL is that working with a proof system based on functors, and taking into account a partial order relation on each type, it is possible to define static characteristics on functors, namely a functor can be either upward monotone, or downward monotone; of course, a functor can also be non-monotone in the following terms:

Definition 2.2.2. [Dow94] A function $d \in D_{(\alpha,\beta)}$ is:

1. *upward monotone* (+d) if and only if

$$\text{for all } x, y \in D_\alpha, x \leq_\alpha y \text{ implies that } d(x) \leq_\beta d(y), \tag{2.1}$$

2. *downward monotone* (−d) if and only if

$$\text{for all } x, y \in D_\alpha, x \leq_\alpha y \text{ implies that } d(y) \leq_\beta d(x). \tag{2.2}$$

3. *non-monotone* (·d) if and only if it is neither upward monotone, nor downward monotone. □

The inference rule $\frac{(\alpha,\beta) \quad \alpha}{\beta}$ must be applied to construct a proof tree, therefore a functor node and an argument node are required. The resulting node will serve as either the functor node, or the argument node to construct the following level of the proof tree. In this way, the construction of a proof tree is done by composing functors: if an upward (downward) monotone functor α is argument of a functor β, then the static characteristic of α can change, depending of the static characteristic of β. In Table 2.2 [IIM14] the result of composing upward monotone (+), downward monotone (−), and non-monotone (·) functors is shown.

∘	+	−	·
+	+	−	·
−	−	+	·
·	·	·	·

Table 2.2: Result of the composition of upward monotone (+), downward monotone (−), and non-monotone (·) functors.

From Table 2.2 we can infer that the composition of m upward and downward monotone functors will be upward monotone if the number of downward monotone functors is even, otherwise the composition will be downward monotone, and if one of the functors in composition is non-monotone, then the whole composition will be non-monotone. The composition of functors is called *polarity*, it is said that polarity is positive (+) when the composition is upward monotone, negative (−) when the composition is downward monotone, and neutral (·) when the composition is non-monotone. Hence, polarity is a dynamic characteristic of some functors, which is given by the position of the functors in the composition.

In terms of the proof tree of a sentence, a functor node that is upward monotone will have positive polarity if it is the argument of a composition where an even number of downward monotone functors are involved, it will have negative polarity if it is the argument of a composition where an odd number of downward monotone functors are involved, otherwise it will not have polarity. On the same terms, a functor node that is downward monotone will have positive polarity if it is the argument of a composition where an odd number of downward monotone functors are involved, it will have negative polarity if it is the argument of a composition where an even number of downward monotone functors are involved, otherwise it will not have polarity. Finally, a non-monotone functor will not have polarity.

Once the polarity of a node i is known in the proof tree of the sentence S, the subtree whose root is node i can be replaced for a greater one (in the sense of Definition 2.2.1) if node i has positive polarity, giving as a result sentence S'; in a dual way, the subtree whose root is node i can be replaced for a lesser one (in the sense of Definition 2.2.1) if node i has negative polarity, giving as a result sentence S'. In both cases, it is said that S implies S'.

When we say that $N(M)$ is a natural language expression, we also mean that it has M as subexpression; a formal definition of *subexpression* will be given later.

Definition 2.2.3. [Dow94] *Implication in the Proof Theory.* Let $N(M)$, and $N(M')$ be two natural language expressions with $M, M' : \alpha, M \neq M'$, we define that $N(M)$ *implies in the proof theory of NL* $N(M')$ (symbolically $N(M) \vdash N(M')$) in the following way:

$$N(M) \vdash N(M') \text{ if and only if } \begin{cases} M \text{ has positive polarity and } M \leq_\alpha M', \text{ or} \\ M \text{ has negative polarity and } M' \leq_\alpha M. \end{cases}$$

□

Sánchez[SV91] proved the soundness of the above implication with respect to the semantics associated with his proof theory. This is the way of reasoning in NL.

For example, given the sentence *Dobie didn't bring every ball*, if we accept that in this sentence *ball* has polarity $-$, and $\mathtt{big\ ball} \leq_\alpha \mathtt{ball}$, then *Dobie didn't bring every ball* entails *Dobie didn't bring every big ball*.

Sánchez [SV91] created an algorithm to compute polarity based on the algorithm of van Benthem [vB86, vB91] for the same purpose, but Sánchez did not prove their equivalence.

Dowty [Dow94] proposed an internal algorithm for polarity marking, his proposal is more elegant than Sánchez's because, in building the proof tree, the proof system

assigns the right polarity to each node in the tree. Dowty supposed that his algorithm is equivalent to the algorithm of Sánchez, but he had no proof. He did not prove the soundness of his proposal with respect to some semantics, nor did he justify formally why some nodes belong to two different categories.

van Eijck [vE07] defined an external algorithm for polarity marking. His algorithm is interesting because he avoided categories from having monotonicity marks for polarity marking, instead he decorated the resulting category of each functor with the so called polarity mappings; the mapping i designates that polarity is kept, the mapping r indicates that polarity is reversed; and the mapping b designates that polarity is broken. He did not prove that his algorithm is equivalent or better, in some sense, to previous ones, nor did he offer a proof of the soundness of his algorithm.

Moss [Mos12] formalized the proposal for internal polarity marking of Dowty by a theory for monotonic functions through preorders (for monotone increasing functions) and its opposites (for monotone decreasing functions). Specifically, an increasing monotonic function is a monotonic function from the preorder defined over its domain to the preorder defined over its range, and a decreasing monotonic function is a monotonic function from the preorder defined over its domain to the opposite preorder defined over its range. His theory allows him to prove that types σ^+ and σ^- are isomorphic, with this, it is formalized that some nodes in the proof tree have two types. Also, he states the soundness of internalized polarity marking with respect to certain semantics, through the Context Lemma.

2.3 Solving Anaphora

Anaphora is the linguistic phenomenon in which a reference points back to an entity previously introduced in discourse. The reference is called anaphor and the pointed entity is called antecedent. Anaphora resolution is very important because it guarantees the cohesion among the sentences of a discourse.

Mitkov [Mit02] identifies the following kinds of anaphora according to the form of the anaphor:

- pronominal anaphora, when the anaphor is a personal, possessive, reflexive, demonstrative, or relative pronoun. *Mary loves a book which Smith abhors*.
- lexical noun phrase anaphora, the anaphor is a definite noun phrase or a proper name. *Both noses went down to the footprints in the snow. These footprints were very fresh*.

- noun anaphora, the anaphor is a non-lexical proform pointing back to the head noun or nominal group of a noun phrase. *I don't think I'll have a sweet pretzel, just a plain one*.
- verb anaphora, the auxiliary verb do (in any conjugation) is the anaphor that points back to a verb or verb phrase. *Romeo Dallaire, the Canadian general in charge, begged for reinforcements; so did Boutros-Ghali*.
- adverb anaphora, the anaphor can be a locative adverb as *there*, or a temporal adverb as *then*. *Will you walk with me to the garden? I've got to go down there and Bugs has to go to the longhouse*.

According to the locations of the anaphor and the antecedent there are only two kinds of anaphora, intrasentential (sentence) anaphora and intersentential (discourse) anaphora. Therefore, the previous examples of pronominal and noun anaphora are intrasentential, the others are intersentencial.

From the very definition of anaphora, a general algorithm to solve anaphora can be easily inferred: 1. identification of anaphors, 2. location of potencial antecedents, 3. resolution of anaphora, by choosing, for each anaphor identified, an antecedent from the set of potencial antecedents.

In solving anaphora the following kind of knowledge is required:

- morphological and lexical, in identifying pronouns, and in knowing the gender and number of nouns and pronouns.
- syntactic, to identify proper nouns, common nouns and noun phrases.
- semantic, consider the example: *The petrified kitten refused to come down from the tree. It gazed beseechingly at the onlookers below.* It is not possible to decide if *it* is pointing back to *The petrified kitten* or *the tree* because both are singular and gender neutral. To choose one of them, it is necessary the semantic knowledge about the verb *gaze* which requieres that the subject in an active voice sentence be animate, and also that kitten is an animated entity.
- discourse, consider the example: *Tilly tried on the dress over her skirt and ripped it.* Without knowing the context it is not possible to assure if *it* refers to *the dress* or to *the skirt*.
- real world or common sense, consider the example: *The soldiers shot at the women and they fell.* Only having the rule "If X shoots at Y and if Z ($Z \in \{X, Y\}$) falls, then it is more likely for Z to be Y", it can be said that *they* refers to *the women*.

From the different kinds of anaphora, pronominal anaphora is the most usually found, and also it is the easiest to solve, because the set of pronouns is finite and is clearly identified[1]. Compare this with the case of lexical noun phrase anaphora, actually it is very hard to compute when a definite noun phrase is playing the role of an anaphor.

Paradigms of anaphora resolution in the 1960s, 1970s and 1980s were mainly based on some of the knowledge already mentioned, it is said that they are methods of extensive domain and linguistic knowledge.

From the 1990s up to now, the paradigm shifts to knowledge poor methods, this was possible because the availability of Part of Speech Taggers, shallow parsers, raw and annotated corpora, for example. With these tools, methods for solving anaphora, based on probability, clustering and machine learning techniques, have been created.

But, also, logical formalisms have been used, the work of Montague about logical semantics of natural language [Mon70a, Mon70b, Mon73] has been the inspiration of almost all the subsequent research on formal syntax and semantics, both to criticize it and to improve it. He based his work on predicate logic and lambda calculus. His main insight is that the semantics of a sentence depends only on its constituent parts, the so called principle of compositionality.

The principle of compositionality fails with the intersentential anaphora, because the antecedent of an anaphor is in a previous sentence. Hence, the meaning of a sentence change in accordance to its context. In this sense the meaning of a sentence is more dynamic than static. That is the reason to present two of the more cited works about pronominal anaphora resolution.

Both of them end representing the sentences of a discourse in First Order Logic and have a model theoretic semantics, but Discourse Representation Theory has the advantage of translating sentences in natural language to first order formulas, in an automatic fashion.

2.3.1 Discourse Representation Theory

Discourse Representation Theory (DRT) of Kamp and Reyle [Kam81, KR93, KVGR11] is considered the first formal proposal that solves intersentential anaphora. Indefinite noun phrases introduce new free variables, called discourse referentes (also the constants

[1] In Computational Linguistics it is almost impossible to talk of general rules, any try of generalization sooner or later is reached by counterexamples. Of course, anaphora resolution is not the exception. Such is the case of the so called pleonastic it, in which it does not appear in a referential way, as in the sentence *It is late.*

associated with proper names are considered discourse referents), definite noun phrases and pronouns are also variables, but they are linked to appropiate antecedent variables.

An intransitive verb is treated as a relation constant of arity one, and if it is applied to a discourse referent forms a condition; transitive verbs are relation constants of arity two, and if they are applied to two discourse referents they are also a condition. The appropiate linking of the discourse referent δ_1 to the previous one δ_2, δ_1 **is** δ_2 is also a condition.

The main representation structure in DRT is called a box, this box has two parts and it is written $[x_1 \dots x_m | \gamma_1, \dots, \gamma_n]$, where $x_1, \dots, x_m, m \geq 0$ are variables, and $\gamma_1, \dots, \gamma_n (n \geq 0)$ are conditions. It is supposed that each processed sentence produces a box, but in accordance with the kind of discourse the boxes newly created are composed of previous boxes by means of the logical connectives **not, or** and $\Rightarrow$, creating compound conditions.

After processing all the sentences of a discourse, a compound box is obtained and it is interpreted on ordinary first-order models. In other words, on ending the analysis of discourse a logical form in first order logic is obtained.

In the interpretation part, the dynamizer concept is *embedding*, it is a function mapping discourse referent variables into elements in the domain. Hence, in processing a sentence, the previous embedding changes in accordance to the new discourse referents introduced by the sentence being processed, nothing more has to change.

Without going into details, consider the following discourse:

(2.3) A man walks in the park. He whistles

The box corresponding to (2.3) is:

$$[x | man(x) \wedge walk\_in\_the\_park(x) \wedge whistle(x)]$$

Which means that, the discourse referent x is in the scope of the conditions $man(x)$, $walk\_in\_the\_park(x)$, and $whistle(x)$. Solving in this way the problem of intersentential anaphora.

2.3.2 Dynamic Predicate Logic

Dynamic Predicate Logic (DPL) is the proposal of Groenendijk and Stokhof [GS91] to recover the principle of compositionality in solving intersentential anaphora. Consider again the pair of sentences in 2.3

The interest here is that the pronoun "He" in the second sentence be bound to the indefinite noun phrase "A man" of the first sentence. As an indefinite noun phrase

is associated with an existential quantifier and an appropriate predicate, a possible translation of (2.3) in Predicate Logic (PL) is:

(2.4) $\exists x[man(x) \wedge walk\_in\_the\_park(x) \wedge whistle(x)]$

But, translating the first sentence of (2.3) to PL, $\exists x[man(x) \wedge walk\_in\_the\_park(x)]$ is gotten and this is not a subformula of (2.4).

If it is wished that the representations of the sentences of a discourse be subformulas of the representation of the discourse as a whole, then the following translation of (2.3) is needed.

(2.5) $\exists x[man(x) \wedge walk\_in\_the\_park(x)] \wedge whistle(x)$

Unfortunately, in this case the variable x in $whistle(x)$ is not in the scope of the existential quantifier. Notice that the concatenation of the sentences in a discourse has been translated to predicate formulas through the conjunction of formulas.

It should be noted that previous to each sentence there is information, that each sentence may contribute with more information, and that all the information collected has to be passed to the following sentence.

Information in DPL is intended as a function mapping variables to elements in the domain of discourse, such a function is called an assigment. If G is the set of all asignments, then $[\![]\!]^{DPL} \subseteq G \times G$. Intuitively, it means that each formula has input information and output information, depending on the kind of formula, input information may change or not.

If the assignment g is the input information and the assignment h is the output information for $\phi \wedge \psi$, then g will be the input information for ϕ that may change it to get the output information k; now k will be the input information for ψ that may change it to the output information h, in this way the information can be passed from a sentence to the next one, formally:

$$[\![\phi \wedge \psi]\!] = \{\langle g, h\rangle \mid \exists k : \langle g, k\rangle \in [\![\phi]\!] \;\&\; \langle k, h\rangle \in [\![\psi]\!]\} \tag{2.6}$$

If the assignment g is the input information and the assignment h is the output information for $\exists x\phi$, then an assignment k, which is as g except in x, should exist, the assingnment k will be the input information for the formula ϕ and it may change it to get the output information h. Hence the existential quantifier is the unique constructor of formulas that changes the input information, this is formally defined by:

$$[\![\exists x\phi]\!] = \{\langle g, h\rangle \mid \exists k : k[x]g \;\&\; \langle k, h\rangle \in [\![\phi]\!]\} \tag{2.7}$$

The meaning of the other constructors for formulas is the same as in PL, because they do not change the input information, which is denoted by $h = g$ in its respective definitions. The definition of the meaning of an atomic predicate is presented in order to follow the example 2.3.1.

$$[\![Rt_1 \dots t_n]\!] = \{\langle g, h\rangle \mid h = g \;\&\; \langle [\![t_1]\!]_h, \dots, [\![t_n]\!]_h\rangle \in F(R)\} \tag{2.8}$$

Example 2.3.1. In DPL $[\![\exists x[man(x) \wedge walk\_in\_the\_park(x)] \wedge whistle(x)]\!] = [\![\exists x[man(x) \wedge walk\_in\_the\_park(x) \wedge whistle(x)]]\!]$.

$$[\![\exists x[man(x) \wedge walk\_in\_the\_park(x)] \wedge whistle(x)]\!] \overset{2.6}{=}$$

$$\{\langle g, h\rangle \mid \exists k : \langle g, k\rangle \in [\![\exists x[man(x) \wedge walk\_in\_the\_park(x)]]\!] \;\&\; \langle k, h\rangle \in [\![whistle(x)]\!]\} \overset{2.7}{=}$$

$$\{\langle g, h\rangle \mid \exists k : \exists k' : k'[x]g \;\&\; \langle k', k\rangle \in [\![man(x) \wedge walk\_in\_the\_park(x)]\!]$$
$$\&\; \langle k, h\rangle \in [\![whistle(x)]\!]\} \overset{2.6}{=}$$

$$\{\langle g, h\rangle \mid \exists k : \exists k' : k'[x]g \; \exists k'' : \langle k', k''\rangle \in [\![man(x)]\!] \;\&\; \langle k'', k\rangle \in [\![walk\_in\_the\_park(x)]\!]$$
$$\&\; \langle k, h\rangle \in [\![whistle(x)]\!]\} \overset{2.8}{=}$$

$$\{\langle g, h\rangle \mid \exists k : \exists k' : k'[x]g \; \exists k'' : k'' = k' \;\&\; k''(x) \in F(man) \;\&\; \langle k'', k\rangle \in [\![walk\_in\_the\_park(x)]\!]$$
$$\&\; \langle k, h\rangle \in [\![whistle(x)]\!]\} \overset{2.8}{=}$$

$$\{\langle g, h\rangle \mid \exists k : \exists k' : k'[x]g \;\&\; k'(x) \in F(man) \;\&\; k = k' \;\&\; k(x) \in F(walk\_in\_the\_park)$$
$$\&\; \langle k, h\rangle \in [\![whistle(x)]\!]\} \overset{2.8}{=}$$

$$\{\langle g, h\rangle \mid \exists k' : k'[x]g \;\&\; k'(x) \in F(man) \;\&\; k'(x) \in F(walk\_in\_the\_park)$$
$$\&\; h = k' \;\&\; h(x) \in F(whistle)\} =$$

$$\{\langle g, h\rangle \mid \exists h : h[x]g \;\&\; h(x) \in F(man) \;\&\; h(x) \in F(walk\_in\_the\_park) \;\&\; h(x) \in F(whistle)\} \overset{2.7}{=}$$

$$[\![\exists x[man(x) \wedge walk\_in\_the\_park(x) \wedge whistle(x)]]\!]$$

It should be mentioned that there exist proposals to dynamise higher order logics [GS89], [Eij99], [Jäg05], [dG06], [MP14], but they are out of the scope of this work.

Chapter 3

Polarity Algorithms

In this chapter the general concept of polarity is discussed in some detail. Later, the algorithms of van Benthem [vB86, vB91], Sánchez [SV91], Dowty [Dow94], and van Eijck [vE07] are presented and explained.

Finally, the equivalence among the four algorithms is proved, which makes clear that internal and external polarity marking compute the same notion of polarity. As a further consequence, the soundness of the algorithms of van Benthem and van Eijck is established.

3.1 Introduction

The concept of polarity is pervasive in natural language, relating narrowly syntax, semantics and pragmatics [Gia11, Isr11], it refers to items of many syntactic categories such as nouns, verbs and adverbs. *Neutral polarity items* are those that can appear in affirmative and negative sentences, *negative polarity items* (NPI) cannot appear in affirmative sentences, and *positive polarity items* (PPI) cannot appear in negative sentences.

If polarity constraints are not fulfilled for NPIs and PPIs, the sentence is ill-formed, not as a consequence of a syntactic rule, but because of its meaninglessness. For example [Gia11]: *yet* is a NPI, so it is right to write *Bill isn't here yet*, but it is wrong to write *Bill is here yet*; *already* is a PPI, then it is right to write *John is here already*, but *John isn't here already* is not.

The main matter is to characterize what expressions licence NPIs and PPIs. According to Israel [Isr11], Ladusaw found that most of NPIs in English are licenced by *downward entailing propositional operators*, these operators reverse the relation of semantic strength among expressions. Similarly, PPIs are licenced by *upward entailing propositional operators*, they preserve the relation of semantic strength among expressions.

Expression e_1 is semantically stronger than expression e_2 if the set denoted by the meaning of e_1 is a subset of the set denoted by the meaning of e_2; hence, in accordance

with Ladusaw (quoted in [Isr11]) to find a grammar for polarity items it is necessary to use some logical form for sentences, and then their representation in a model theory.

An alternative from logic to the Ladusaw proposal is NL. Although NL goes back to Aristotle, the core idea of the NL Program of van Benthem et al. [vB07, Kar15] is that natural language, besides being a vehicle to communicate things, also serves to reason without having to use some logical form as predicate logic or high order logic; i.e., the syntactic analysis of a sentence is enough to reason, avoiding a model theory.

NL uses a proof theory, called categorial grammar, to parse a sentence [vB86, vB91, SV91]. In these grammars each word in a sentence is assigned to a type (also called category). It is assumed that types denote sets, each denotation is partially ordered, and also that types behave as functors. Some functors are upward monotone and correspond to the Ladusaw upward entailing propositional operators, others are downward monotone and correspond to the Ladusaw downward entailing propositional operators, some others are not monotone. Thus, characterizing functors according to their monotonicity is a static feature.

In order to know if an expression is a well formed sentence, the functors, which are assigned to each word in the expression, are composed forming a proof tree. Monotonicity of functors could change according to their place in composition; this is the concept of polarity in NL, it is a dynamic feature that allows us to reason, a generalization of the monotonicity concept.

Remembering that the denotations of types are partially ordered, if on parsing a sentence one of their expressions has positive polarity, then it can be replaced with a greater expression; if the expression has negative polarity, then it can be replaced with a lesser expression, without changing in both cases the truth value of the sentences.

We want to characterize the possibilities of NL in problems that require textual inference, and we have found different algorithms for polarity marking. Specifically, we refer to the algorithms of van Benthem [vB86, vB91], Sánchez [SV91], Dowty [Dow94], and van Eijck [vE07].

In his paper, Dowty supposes that his algorithm is equivalent to the algorithm of Sánchez, nevertheless, he does not prove it. Moss [Mos12] proves that Dowty's internal algorithm is sound, but neither its equivalence with an external algorithm has been proved, nor what the relation between the semantics of Moss and the semantics of Sánchez is. Finally, Sánchez is recognized because he was the one who formalized van Benthem's ideas about NL.

In a few words, the following questions arise:

- Is the polarity computed by the external algorithm of Sánchez equal to the polarity

computed by the internal algorithm of Dowty?

- Do the external algorithms of van Benthem and van Eijck compute the same polarity as that of the external algorithm of Sánchez?
- What is the relation between the semantics of Sánchez and the semantics of Moss?

We decided to prove the equivalence between the four analyzed algorithms, showing by this means that internal and external algorithms for polarity marking are equivalent, and collaterally that the algorithms of van Benthem and van Eijck are sound.

Even more, as the algorithms of Sánchez and Dowty are equivalent, we have that the semantics proposed by Sánchez, for his algorithm for polarity marking, is equivalent to the semantics proposed by Moss for the algorithm of Dowty. This is because independently of how each semantics justifies the soundness of computing polarity by the respective algorithm, once the equivalence between the two algorithms is established, the two semantics make sound each way to compute polarity; in this sense the semantics of Sánchez and Dowty are equivalent.

3.2 Polarity Marking

There are two sorts of polarity marking algorithms: external and internal. The external ones compute the polarity outside the proof system, nevertheless they use the proof tree to compute polarity. The internal methods compute polarity at the same time that the proof tree is being constructed, i.e., the proof system is designed to mark each node in the proof tree with its corresponding polarity.

In this part, we discuss the algorithms for polarity marking of van Benthem, Sánchez, Dowty and van Eijck.

The algorithm of van Benthem constructs the proof tree from the leaves to the root. To assign polarity marks to the nodes of the tree, it proceeds from the root to the leaves. The algorithm of Sánchez constructs the proof tree from the leaves to the root, assigning polarity marks is also done from the leaves to the root. In the algorithm of Dowty, polarity marks are computed at the same time as the proof tree is being built from the root to the leaves. The polarity map assignment algorithm of van Eijck, is done from the leaves to the root, polarity marks are computed from the root to the leaves.

3.2.1 van Benthem's Algorithm

According to Icard and Moss [IIM14], van Benthem [vB86] and Sánchez [SV91] define proof systems to reason about entailment using monotonicity in higher order languages.

Both van Benthem and Sánchez use, for the syntactic analysis of a sentence, a version of categorial grammars due to Ajdukiewicz. This is based on the basic types e (for entities), and t (for truth values), more complex types of the categorial language are constructed recursively by the creation of functors, formally:

Definition 3.2.1. [Dow94] The categorial language of the calculus of Ajdukiewicz $\mathcal{LL}$ is given by:

1. e and t belong to $\mathcal{LL}$.
2. If α and β belong to $\mathcal{LL}$, then (α, β) also belongs to $\mathcal{LL}$.

If (α, β) in $\mathcal{LL}$, then the unique inference rule in the calculus of Ajdukiewicz takes one of the forms:

$$\frac{(\alpha,\beta) \quad \alpha}{\beta} \text{ or } \frac{\alpha \quad (\alpha,\beta)}{\beta}, \tag{3.1}$$

i.e., it does not matter if the type α appears either on the left, or on the right of the functor (α, β). □

We have a proof tree in Figure 3.1 for the sentence *Dobie didn't bring every ball*. In this figure, it is worth noting that: the type of *every* has its first argument on the right; the type of *every ball* has its argument on the left; the type of *Dobie* has its argument on the right, and the type t has been derived from both of them, indicating that the expression in natural language is a well formed sentence.

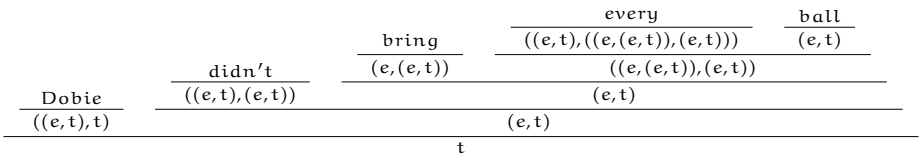


Figure 3.1: Proof tree for the sentence *Dobie didn't bring every ball*.

In order to mark either the upward monotonicity of a functor from α to β, (α^+, β), or the downward monotonicity of a functor from α to β, (α^-, β), the following clause is added to DEFINITION 3.2.1:

$$\text{If } (\alpha,\beta) \text{ is in } \mathcal{LL}, \text{ then } (\alpha^+,\beta), \text{ and } (\alpha^-,\beta) \text{ are also in } \mathcal{LL}.$$

So that, it is supposed that the lexicon contains the information of monotonicity that some functors require, as shown below:

$$\texttt{Dobie} : ((e,t)^{+},t) \qquad \texttt{didn't} : ((e,t)^{-},(e,t))$$
$$\texttt{bring} : (e,(e,t)) \qquad \textit{every} : ((e,t)^{-},((e,(e,t))^{+},(e,t)))$$
$$\texttt{ball} : (e,t)$$

As an example, we have the proof tree in Figure 3.2 with lexical monotonicity marking for the sentence *Dobie didn't bring every ball*.

$$\dfrac{\dfrac{\texttt{Dobie}}{((e,t)^{+},t)} \qquad \dfrac{\dfrac{\texttt{didn't}}{((e,t)^{-},(e,t))} \qquad \dfrac{\dfrac{\texttt{bring}}{(e,(e,t))} \qquad \dfrac{\dfrac{\textit{every}}{((e,t)^{-},((e,(e,t))^{+},(e,t)))} \qquad \dfrac{\texttt{ball}}{(e,t)}}{((e,(e,t))^{+},(e,t))}}{(e,t)}}{(e,t)}}{t}$$

Figure 3.2: Proof tree with lexical monotonicity marking for the sentence *Dobie didn't bring every ball*.

Once the proof tree has lexical monotonicity marks, the algorithm of van Benthem begins marking the root of the proof tree with polarity +, then if the functor in turn is upward monotone, the polarity mark is propagated; in case that the functor is downward monotone, then the polarity mark of the argument is reversed because a downward monotone functor reverses the order relation of the elements of its domain.

Algorithm 3.2.2. van Benthem external polarity algorithm.

1. Label the root with +.
2. Propagate notations up the tree.
   (a) If a node of type β is labeled l and its children are of type (α^{+},β) and α, then both children are labeled l (diagrammatically, $\dfrac{(\alpha^{+},\beta)_{l} \quad \alpha_{l}}{\beta_{l}}$).
   (b) If a node of type β is labeled l and its children are of type (α^{-},β) and α, then the former child is to be labeled l and the latter child is to be labeled $-l$, that is, the flipped version of l (diagrammatically, $\dfrac{(\alpha^{-},\beta)_{l} \quad \alpha_{-l}}{\beta_{l}}$).

Figure 3.3 exemplifies Algorithm 3.2.2 for the sentence *Dobie didn't bring every ball*.

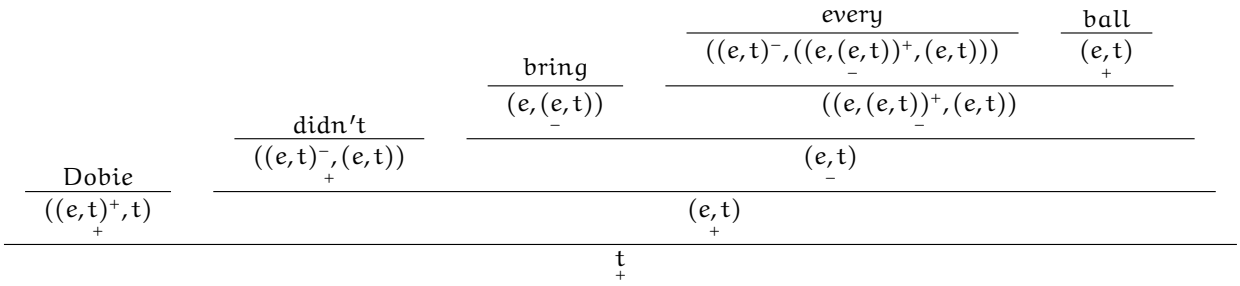


Figure 3.3: Proof tree with polarity marks for the sentence *Dobie didn't bring every ball*, using the algorithm of van Benthem.

3.2.2 Sánchez's Algorithm

As stated by Dowty [Dow94], to find the polarity of the elements of a sentence, Sánchez defines a three step method: lexical monotonicity marking, external monotonicity marking, and determining the polarity of each constituent.

For the lexical monotonicity marking, he does the same as van Benthem, i.e., he considers that the lexicon has information about the sort of monotonicity of some functors. Figure 3.2 shows a proof tree with lexical marks for *Dobie didn't bring every ball*.

For the external monotonicity marking, he adds "+" and "−" to the functor and argument nodes in a proof tree, according to the following rules:

$$\frac{(\alpha,\beta) \qquad \alpha}{\beta} \Rightarrow \frac{\underset{+}{(\alpha,\beta)} \qquad \alpha}{\beta}, \tag{3.2}$$

$$\frac{(\alpha^{+},\beta) \qquad \alpha}{\beta} \Rightarrow \frac{\underset{+}{(\alpha^{+},\beta)} \qquad \underset{+}{\alpha}}{\beta}, \tag{3.3}$$

$$\frac{(\alpha^{-},\beta) \qquad \alpha}{\beta} \Rightarrow \frac{\underset{+}{(\alpha^{-},\beta)} \qquad \underset{-}{\alpha}}{\beta}, \tag{3.4}$$

A proof tree with external monotonicity marks is shown in Figure 3.4 for the sentence *Dobie didn't bring every ball*.

Once having the proof tree with external monotonicity marks, Sánchez defines the following algorithm to compute polarity.

Algorithm 3.2.3. Sánchez external polarity algorithm. If D is a proof tree with external monotonicity marks, and root node ρ, then:

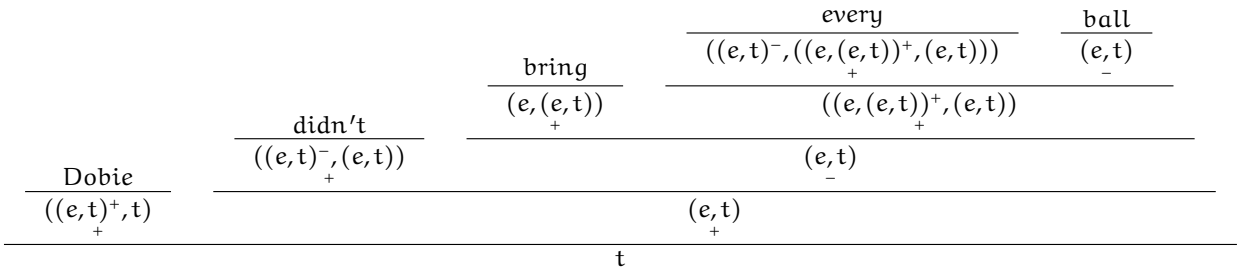


Figure 3.4: Proof tree with external monotonicity marks for the sentence *Dobie didn't bring every ball*.

1. a node γ has polarity in D if and only if each node on the path from γ to ρ has external monotonicity mark, and

2. a node γ is *positive* if and only if γ has polarity and if the number of nodes in this path marked by "−" is even;

3. a node γ is *negative* if and only if γ has polarity and if the number of nodes in this path marked by "−" is odd;

On applying the algorithm of Sánchez to the proof tree with external monotonicity marks for the sentence *Dobie didn't bring every ball* (Figure 3.4), we get the proof tree with polarity marks of Figure 3.3.

3.2.3 Dowty's Algorithm

Dowty [Dow94] defines an internal algorithm, in which marking of monotonicity and computing the polarity are made in only one step, when the proof tree is being built. So polarity determination is done within the proof system, this is why it is called internal.

Dowty uses a version of categorial grammars that takes into account the position of the argument of each functor, hence to form a new functor category there are two operators "/" and "\". Intuitively, A/B(B\A) is a functor category that search for B on the right (left) to give A as result.

Definition 3.2.4. [Dow94] The categorial language is the following:

1. NP (= type e), S (= type t), and CN (= (e,t)) are primitive categories.

2. If A and B are any categories, so are (A/B)[1] and $(B\backslash A)$.

3. If (A/B) is a category, then we have that (A^+/B^+), (A^+/B^-), (A^-/B^+) and (A^-/B^-) are also categories.

4. If $(B\backslash A)$ is a category, then we have that $(B^+\backslash A^+)$, $(B^-\backslash A^+)$, $(B^+\backslash A^-)$ and $(B^-\backslash A^-)$ are also categories. □

As it has been seen, in the proposals of van Benthem and Sánchez the symbols "+" and "−" are used for two related purposes: to indicate if the functors are monotone either upward, or downward, and also to indicate if the polarity is either positive, or negative. In the proposal of Dowty these symbols only indicate the polarity of each node of the proof tree. To achieve it, the words can belong to two different types, depending on the context in which they appear. For example, *every* can belong to either the category $(TV^+\backslash VP^+)/CN^-$, or to the category $(TV^-\backslash VP^-)/CN^+$, both categories encode the same information, namely that *every* reverses polarity in its first argument, and preserves polarity in its second argument.

We have said that in the proposal of Dowty, the symbols "+" and "−" only indicate polarity, therefore the elements of the lexicon can have both polarity marks "+" and "−". The information, that a functor encodes, refers to: the functors that preserve polarity (for example, (A^+/B^+) and (A^-/B^-)), and the functors that reverse polarity (for example, (A^+/B^-) and (A^-/B^+)).

Definition 3.2.5. [Dow94] Polarity marks and sorts of functors:

1. Lexical items in general appear in both a "+"-marked and a "−"-marked category though with the same interpretation.

2. Polarity preserving functors appear in categories of the form (A^+/B^+), (A^-/B^-), $(B^+\backslash A^+)$, and $(B^-\backslash A^-)$.

3. Polarity reversing functors appear in categories of the form (A^+/B^-), (A^-/B^+), $(B^-\backslash A^+)$, and $(B^+\backslash A^-)$. □

To assign polarity marks to each node, at the same time as the proof tree is constructed, the categorial elimination rules have to be applied according to their character of preserving (reversing) polarity. For example, for preserving polarity there exist rules of the form $\frac{(A^x/B^x) \quad B^x}{A^x}$, with $x \in \{+,-\}$, and for reversing polarity there exist rules of the form $\frac{(A^x/B^y) \quad B^y}{A^x}$, with $x,y \in \{+,-\}, x \neq y$.

[1] Although, in examples where categorial grammars are used with the operators / and \, the most external parentheses will be avoided.

Definition 3.2.6. [Dow94] The categorial elimination rules (or "Functional Application" Rules) must appropiately respect +/− marking:

1. Polarity preserving elimination rules:

$$\frac{(A^+/B^+) \quad B^+}{A^+} \qquad \frac{B^+ \quad (B^+\backslash A^+)}{A^+} \tag{3.5}$$

$$\frac{(A^-/B^-) \quad B^-}{A^-} \qquad \frac{B^- \quad (B^-\backslash A^-)}{A^-} \tag{3.6}$$

2. Polarity reversing elimination rules:

$$\frac{(A^+/B^-) \quad B^-}{A^+} \qquad \frac{B^- \quad (B^-\backslash A^+)}{A^+} \tag{3.7}$$

$$\frac{(A^-/B^+) \quad B^+}{A^-} \qquad \frac{B^+ \quad (B^+\backslash A^-)}{A^-} \tag{3.8}$$

□

$$\dfrac{\dfrac{\text{Dobie}}{S^+/VP^+} \quad \dfrac{\dfrac{\text{didn't}}{VP^+/VP^-} \quad \dfrac{\dfrac{\text{bring}}{TV^-} \quad \dfrac{\dfrac{\text{every}}{(TV^-\backslash VP^-)/CN^+} \quad \dfrac{\text{ball}}{CN^+}}{TV^-\backslash VP^-}}{VP^-}}{VP^+}}{S^+}$$

Figure 3.5: Proof tree with polarity marks using the algorithm of Dowty for the sentence *Dobie didn't bring every ball*.

Now let us construct the proof tree of Figure 3.5. We have to derive S^+: as *Dobie* is a functor that preserves polarity it could belong to the category S^+/VP^+ or S^-/VP^-, but we want to derive S^+, therefore we can only apply the rule for / of (3.5) marking with polarity "+" the argument, i.e., VP^+.

In order to derive VP^+, we note that *didn't* is a functor that reverses polarity, hence it could belong to the categories VP^+/VP^- or VP^-/VP^+, but we want to derive VP^+, so we apply the rule / of (3.7), marking with polarity "−" the argument, i.e., VP^-.

Now we have to derive VP^-, *bring* could belong to the category TV^+ or TV^-, we suppose that it belongs to TV^+, as we wish to derive VP^- we apply the rule for \ of (3.8), now we must derive $TV^+\backslash VP^-$.

To derive $TV^+\backslash VP^-$ we have that *every* reverses polarity in its first argument, and preserves polarity in its second argument, therefore its possible categories are $(TV^+\backslash VP^+)/CN^-$

and $(TV^{-}\backslash VP^{-})/CN^{+}$, we realize here that it is not possible to derive $TV^{+}\backslash VP^{-}$, therefore our supposition that *bring* belonged to TV^{+} is wrong, hence it should belong to TV^{-}, now we have to derive $TV^{-}\backslash VP^{-}$.

In order to derive $TV^{-}\backslash VP^{-}$, we have that *every* reverses polarity in its first argument, and preserves polarity in its second argument, therefore its possible categories are $(TV^{+}\backslash VP^{+})/CN^{-}$ and $(TV^{-}\backslash VP^{-})/CN^{+}$, but we want to derive $TV^{-}\backslash VP^{-}$, so *every* has to belong to the category $(TV^{-}\backslash VP^{-})/CN^{+}$, which also determines that *ball* has to belong to the category CN^{+}, in the last step we use the rule for / of (3.8).

3.2.4 van Eijck's Algorithm

Although, van Eijck does not define formally the categorial language that he used in his work, it is inferred from context that it is a categorial grammar as the one found in the paper of Dowty, except that van Eijck uses a binary functor of the form $A\backslash C/B$, as can be expected, it searches to the left for A, and to the right for B, giving as result C.

To avoid increasing the categorial language at keeping track of the monotonicity of functors (as it is done in the proposals of van Benthem and Sánchez), or their composition (as it is done by Dowty), van Eijck uses three mappings on the polarity marks $m = \{+, -, 0\}$[2]. The domain and range of the three mappings (these are called marker transformers) is m. The first mapping is identity i, which preserves monotonicity; the second mapping is reverse r, which reverses polarity, given for $r(+) = -$, $r(-) = +$ and $r(0) = 0$; the third mapping is break b, given for $b(x) = 0$, which marks non-monotonicity.

The polarity mappings are annotations over the resulting category of the functors, as it is observed in the following example of lexicon.

$$\begin{array}{ll} \mathrm{Dobie} : (S_i/VP) & \mathrm{didn't} : (VP_r/VP) \\ \mathrm{bring} : TV & \mathit{every} : ((TV\backslash VP_i)_r/CN) \\ \mathrm{ball} : CN & \end{array}$$

Later, van Eijck defines the algorithm 3.2.7, this propagates the polarity mappings from the leaves to the root of the tree. First, the categories that only are arguments do not have polarity mapping (for example, *bring*). Then, the polarity mapping of the resulting category of the current functor, is passed on to the category that is obtained by applying the functor to its arguments, whether the arguments have or do not have a polarity mapping assigned. This is done until the root is reached.

[2]Also, with these mappings the meaning of the symbols + and − is now unique, remember that it was not the case in the proposals of van Benthem and Sánchez.

Algorithm 3.2.7. van Eijck's Polarity Map Assignment

Leaf Marking Functional lexical categories have all their result categories labeled with marker transformers. Lexical argument categories get an unlabeled basic category.

Tree Marking From a function $(C_f/A)_{f'}$ and an argument A_g, (or from an argument A_g and a function $(A\backslash C_f)_{f'}$, or from an argument A_g, a function $(A\backslash C_f/B)_{f'}$ and an argument B_h), where f, f', g and h are marker transformers, it is obtained C_f, diagrammatically: $\frac{(C_f/A)_{f'} \quad A_g}{C_f}$ (or $\frac{A_g \quad (A\backslash C_f)_{f'}}{C_f}$, or $\frac{A_g \quad (A\backslash C_f/B)_{f'} \quad B_h}{C_f}$).

For example, in Figure 3.6 we have a proof tree with polarity map assignment for the sentence *Dobie didn't bring every ball*, in this figure we see that from the nodes $((TV\backslash VP_i)_r/CN)$ and CN it is obtained $(TV\backslash VP_i)_r$, which is the resulting category of the functor $((TV\backslash VP_i)_r/CN)$. Note that in this case the argument CN, because it is a leaf and argument node, does not have a polarity map assigned.

Following with Figure 3.6, we note that from the nodes (VP_r/VP) and VP_i the node VP_r is obtained, in this case the argument node has polarity map i, which is ignored. Finally, from the functor node $(TV\backslash VP_i)_r$ and argument node TV, VP_i is derived; note that the polarity map r of the functor is ignored, only the polarity map i of the resulting category is propagated. Nevertheless, the polarity maps now ignored are very important to mark polarity, as it is seen in Algorithm 3.2.8 and related example.

$$\dfrac{\dfrac{\text{Dobie}}{(S_i/VP)} \quad \dfrac{\dfrac{\text{didn't}}{(VP_r/VP)} \quad \dfrac{\dfrac{\text{bring}}{TV} \quad \dfrac{\dfrac{\text{every}}{((TV\backslash VP_i)_r/CN)} \quad \dfrac{\text{ball}}{CN}}{(TV\backslash VP_i)_r}}{VP_i}}{VP_r}}{S_i}$$

Figure 3.6: Proof tree with polarity map assignment for the sentence *Dobie didn't bring every ball*.

Once we have the proof tree with polarity maps assigned, in order to mark the polarity of each node, van Eijck defines the external Algorithm 3.2.8, this algorithm begins assigning the positive polarity mark + to the root. Then, for the remaining nodes, the polarity mark l, of the node obtained by applying a functor to their arguments, is passed to the functor node, but the argument node is assigned with the resulting polarity by applying the polarity map f to l, i.e., $f(l)$.

Algorithm 3.2.8. van Eijck's Polarity Marking Algorithm

Root Marking The main structure C_f to be marked has positive polarity, so it is marked with +: (C_f^+).

Component Marking If a structure C_f has polarity marking l (C_f^l), then:

Leaf Marking If C_f^l is a leaf, then done.

Composite Marking If C_f^l is derived from a function C_f/A and argument A (or an argument A and a function $A\backslash C_f$, or an argument A, a function $A\backslash C_f/B$ and an argument B), then the function gets polarity marking l, and the arguments get polarity marking $f(l)$, where f is the polarity marking map at node C_f^l, diagrammatically $\dfrac{(C_f/A)^l \quad A^{f(l)}}{C_f^l}$ (or $\dfrac{A^{f(l)} \quad (A\backslash C_f)^l}{C_f^l}$, or $\dfrac{A^{f(l)} \quad (A\backslash C_f/B)^l \quad B^{f(l)}}{C_f^l}$).

$$\dfrac{\dfrac{\text{Dobie}}{(S_i/VP)^+} \quad \dfrac{\dfrac{\text{didn't}}{(VP_r/VP)^+} \quad \dfrac{\dfrac{\text{bring}}{TV^-} \quad \dfrac{\dfrac{\text{every}}{((TV\backslash VP_i)_r/CN)^-} \quad \dfrac{\text{ball}}{CN^+}}{(TV\backslash VP_i)_r^-}}{VP_i^-}}{VP_r^+}}{S_i^+}$$

Figure 3.7: Proof tree with polarity marks for the sentence *Dobie didn't bring every ball*, using the algorithm of van Eijck.

In Figure 3.7, we find a proof tree with polarity marks, applying the algorithm 3.2.8 to the proof tree of Figure 3.6, for the sentence *Dobie didn't bring every ball*.

In order to close this section, in Table 3.1 the main features of each algorithm are summarized.

Feature	van Benthem	Sánchez	Dowty	van Eijck
computes polarity	from root to leaves	from leaves to root	from root to leaves	from root to leaves
kind of functor	non-positional	non-positional	positional	positional
kind of algorithm	external	external	internal	external
categories by lexical item	One	One	Two	One
constructs a proof tree	No	No	Yes	No
needs backtracking	No	No	Yes	No

Table 3.1: Main features of the presented polarity algorithms.

3.3 Algorithmic Equivalences

According to Icard and Moss [IIM14], the first to propose an algorithm for polarity marking was van Benthem [vB86], that is why we prove the equivalence of the algorithms of Sánchez, Dowty, and van Eijck with respect to the algorithm of van Benthem.

Intuitively, any algorithm that computes polarity has to take into account the composition of upward monotone and downward monotone functors, as in Table 2.2. Another element to consider is that the algorithms of van Benthem, Dowty and van Eijck assign polarity marks from the root to the leaves, and the root is marked with positive polarity. The algorithm of Sánchez proceeds from the leaves to the root.

Another important consideration is that, in the algorithms that assign polarity marks from the root to the leaves, the polarity of each node (excepting the root) depends on: the polarity of its parent node, the kind of monotonicity of the functor, and whether the node in consideration is a functor or an argument.

In the algorithm of Sánchez the polarity of each node depends on: the external monotonicity mark of the node, and the parity of the external monotonicity marks on the path from the node to the root.

Of course these considerations determine the cases in the following proofs.

3.3.1 van Benthem and Sánchez

Theorem 3.3.1. *If the algorithm of van Benthem marks node* i *with polarity* l*, then the algorithm of Sánchez marks the same node* i *with polarity* l*.*

Proof. The algorithm of van Benthem marks with polarity $l = +$ (or $-$) to node i. We have two cases:

1. The algorithm of van Benthem uses the clause $\dfrac{(\alpha^{+},\beta)_{l} \quad \alpha_{l}}{\beta_{l}}$. Independently of, whether node i is the functor $(\alpha^{+},\beta)_{l}$ or it is the argument α_{l}, its parent should have positive (or negative) polarity. Since this polarity was propagated from the root node (which always is marked with positive polarity), the clause $\dfrac{(\alpha^{-},\beta)_{l} \quad \alpha_{-l}}{\beta_{l}}$ should have been applied $2n$ (or $2n+1$) times. On the other hand, the algorithm of Sánchez would apply the marking rule for upward monotone functions, namely, $\dfrac{(\alpha^{+},\beta) \quad \alpha}{\beta} \Rightarrow \dfrac{(\alpha^{+},\beta)_{+} \quad \alpha_{+}}{\beta}$. Independently of, whether node i is the functor or it is the argument, by hypothesis there are $2n$ (or $2n+1$) applications of downward monotone functions on the path from its parent to the root. Accordingly the

algorithm of Sánchez applies $2n$ (or $2n+1$) times the rule $\frac{(\alpha^-,\beta)\quad \alpha}{\beta} \Rightarrow \frac{(\alpha^-_+,\beta)\quad \alpha^-}{\beta}$, leaving on the path from its parent to the root $2n$ (or $2n+1$) monotonicity marks $-$, hence marking node i with positive (or negative) polarity.

2. The algorithm of van Benthem uses the clause $\frac{(\alpha^-_l,\beta)\quad \alpha_{-l}}{\beta_l}$. There are two cases:

   (a) If node i is the functor (α^-_l,β) its parent should have positive (or negative) polarity. Since this polarity was propagated from the root node (which always is marked with positive polarity), then the clause $\frac{(\alpha^-_l,\beta)\quad \alpha_{-l}}{\beta_l}$ should have been applied $2n$ (or $2n+1$) times. On the other hand, the algorithm of Sánchez would apply the marking rule for downward monotone functions, namely, $\frac{(\alpha^-,\beta)\quad \alpha}{\beta} \Rightarrow \frac{(\alpha^-_+,\beta)\quad \alpha^-}{\beta}$, then as node i is the functor, by hypothesis there are $2n$ (or $2n+1$) applications of downward monotone functions on the path from its parent to the root. Accordingly the algorithm of Sánchez applies $2n$ (or $2n+1$) times the rule $\frac{(\alpha^-,\beta)\quad \alpha}{\beta} \Rightarrow \frac{(\alpha^-_+,\beta)\quad \alpha^-}{\beta}$, leaving on the path from its parent to the root $2n$ (or $2n+1$) monotonicity marks $-$, hence marking node i with positive (or negative) polarity.

   (b) If node i is the argument α_{-l} its parent should have negative (or positive) polarity. Since this polarity was propagated from the root node (which always is marked with positive polarity), the clause $\frac{(\alpha^-_l,\beta)\quad \alpha_{-l}}{\beta_l}$ should have been applied $2n+1$ (or $2n$) times. On the other hand, the algorithm of Sánchez would apply one time the marking rule for downward monotone functions, namely, $\frac{(\alpha^-,\beta)\quad \alpha}{\beta} \Rightarrow \frac{(\alpha^-_+,\beta)\quad \alpha^-}{\beta}$, then as node i is the argument, by hypothesis there are $2n+1$ (or $2n$) applications of downward monotone functions on the path from its parent to the root. Accordingly the algorithm of Sánchez applies $1+2n+1 = 2n+2$ (or $1+2n = 2n+1$) times the rule $\frac{(\alpha^-,\beta)\quad \alpha}{\beta} \Rightarrow \frac{(\alpha^-_+,\beta)\quad \alpha^-}{\beta}$, leaving on the path from its parent to the root $2n+2$ (or $2n+1$) monotonicity marks $-$, hence marking node i with positive (or negative) polarity.

□

Theorem 3.3.2. *If the algorithm of Sánchez marks node i with polarity l, then the algorithm of van Benthem marks the same node i with polarity l.*

Proof. The algorithm of Sánchez marks node i with polarity $l = +$ (or $-$), we have two cases:

1. Node i is a functor node. Every functor node has $+$ as external monotonicity mark, independently of its lexical monotonicity mark. Therefore, there are $2n$ (or $2n+1$) nodes with monotonicity mark $-$, from its parent to the root. This case arises when the rule $\frac{(\alpha^-, \beta) \quad \alpha}{\beta} \Rightarrow \frac{(\underset{+}{\alpha^-}, \beta) \quad \underset{-}{\alpha}}{\beta}$ is applied $2n$ (or $2n+1$) times due to the use of downward monotone functors; for this type of functors the algorithm of van Benthem would use $2n$ (or $2n+1$) times the clause $\frac{(\underset{l}{\alpha^-}, \beta) \quad \underset{-l}{\alpha}}{\underset{l}{\beta}}$ (that reverses the polarity of the argument). As the algorithm of van Benthem begins marking the root with positive polarity, then positive (or negative) polarity (because l was changed $2n$ (or $2n+1$) times) is propagated to the parent of node i. As node i is a functor node, it does not matter if the algorithm of Sánchez uses the rule $\frac{(\alpha^+, \beta) \quad \alpha}{\beta} \Rightarrow \frac{(\underset{+}{\alpha^+}, \beta) \quad \underset{+}{\alpha}}{\beta}$, or the rule $\frac{(\alpha^-, \beta) \quad \alpha}{\beta} \Rightarrow \frac{(\underset{+}{\alpha^-}, \beta) \quad \underset{-}{\alpha}}{\beta}$; the algorithm of van Benthem would use the corresponding rule of $\frac{(\underset{l}{\alpha^+}, \beta) \quad \underset{l}{\alpha}}{\underset{l}{\beta}}$, or $\frac{(\underset{l}{\alpha^-}, \beta) \quad \underset{-l}{\alpha}}{\underset{l}{\beta}}$; inheriting, in any case, the positive (or negative) polarity of its parent.

2. Node i is an argument node. There are two cases:

   (a) Node i has external monotonicity mark $+$, and there are $2n$ (or $2n+1$) nodes with monotonicity mark $-$, from its parent to the root. This case arises when the rule $\frac{(\alpha^-, \beta) \quad \alpha}{\beta} \Rightarrow \frac{(\underset{+}{\alpha^-}, \beta) \quad \underset{-}{\alpha}}{\beta}$ is applied $2n$ (or $2n+1$) times due to the use of downward monotone functors; for this type of functors the algorithm of van Benthem would use $2n$ (or $2n+1$) times the clause $\frac{(\underset{l}{\alpha^-}, \beta) \quad \underset{-l}{\alpha}}{\underset{l}{\beta}}$ (that reverses the polarity of the argument). As the algorithm of van Benthem begins marking the root with positive polarity, then positive (or negative) polarity (because l was changed $2n$ (or $2n+1$) times) is propagated to the parent of node i. Finally, if node i has $+$ as external monotonicity mark, it is because the rule $\frac{(\alpha^+, \beta) \quad \alpha}{\beta} \Rightarrow \frac{(\underset{+}{\alpha^+}, \beta) \quad \underset{+}{\alpha}}{\beta}$ was applied, hence, the van Benthem's algorithm would apply the clause $\frac{(\underset{l}{\alpha^+}, \beta) \quad \underset{l}{\alpha}}{\underset{l}{\beta}}$, inheriting the positive (or negative) polarity of its parent.

   (b) Node i has external monotonicity mark $-$, and there are $2n+1$ (or $2n$) nodes with monotonicity mark $-$, from its parent to the root. This case arises when

the rule $\frac{(\alpha^-,\beta)\quad \alpha}{\beta} \Rightarrow \frac{(\alpha^-,\beta)\quad \alpha}{\beta}$ is applied $2n+1$ (or $2n$) times due to the use of downward monotone functors; for this type of functors the algorithm of van Benthem would use $2n+1$ (or $2n$) times the clause $\frac{(\alpha^-,\beta)\quad \alpha}{\beta}$ (that reverses the polarity of the argument). As the algorithm of van Benthem begins marking the root with positive polarity, then positive (or negative) polarity (because l was changed $2n$ (or $2n+1$) times) is propagated to the parent of node i. Finally, if node i has $-$ as external monotonicity mark, it is because the rule $\frac{(\alpha^-,\beta)\quad \alpha}{\beta} \Rightarrow \frac{(\alpha^+,\beta)\quad \alpha}{\beta}$ was applied, hence, the van Benthem's algorithm would apply the clause $\frac{(\alpha^-,\beta)\quad \alpha}{\beta}$, reversing from negative (or positive) to positive (or negative) the polarity of node i.

□

3.3.2 van Benthem and Dowty

Theorem 3.3.3. *A node with type β is labeled with polarity l by algorithm 3.2.2 if and only if using the Functional Application Rules of Definition 3.2.6 that node belongs to a type of the form A^l.*

Proof. If algorithm 3.2.2 labeled with polarity $l = +$ (or $-$) the node with type β, then we have two cases:

1. If the children of the node have types (α^+,β) and α, then the polarities of the children and the parent will be $\frac{(\alpha^+,\beta)\quad \alpha}{\beta}$ (or $\frac{(\alpha^+,\beta)\quad \alpha}{\beta}$), which is in biunivocal correspondence with the internal polarity algorithm where the types of the functor and the argument (independently of whether the argument is on the right, or on the left of the functor type) are $\frac{(A^+/B^+)\quad B^+}{A^+}$ or $\frac{B^+\quad (B^+\backslash A^+)}{A^+}$ ($\frac{(A^-/B^-)\quad B^-}{A^-}$ or $\frac{B^-\quad (B^-\backslash A^-)}{A^-}$), i.e., the functor belongs to a type with positive (or negative) polarity and the argument is also in a type with positive (or negative) polarity.

2. If the children of the node have types (α^-,β) and α, then the polarities of the children and the parent will be $\frac{(\alpha^-,\beta)\quad \alpha}{\beta}$ (or $\frac{(\alpha^-,\beta)\quad \alpha}{\beta}$), which is in biunivocal correspondence with the internal polarity algorithm where the types of the functor and the argument (independently of whether the argument is on the right, or on the left of the functor type) are $\frac{(A^+/B^-)\quad B^-}{A^+}$ or $\frac{B^-\quad (B^-\backslash A^+)}{A^+}$ ($\frac{(A^-/B^+)\quad B^+}{A^-}$ or

$\frac{B^{+} \quad (B^{+}\backslash A^{-})}{A^{-}}$), i.e., the functor belongs to a type with positive (or negative) polarity and the argument is in a type with negative (or positive) polarity.

□

3.3.3 van Benthem and van Eijck

Theorem 3.3.4. *The algorithm of van Eijck marks node* i *with polarity* l *if and only if the algorithm of van Benthem marks the same node* i *with polarity* l*.*

Proof. Let us remember that both the algorithm of van Eijck and the algorithm of van Benthem begin assigning polarity + to the root. In general, we have two cases:

1. The functor is unary, we also have two cases:

   (a) The functor is upward monotone. The algorithm of van Eijck uses the rule for the identity map $\frac{(C_i/A)^l \quad A^{i(l)}}{C_i^l}$ (or $\frac{A^{i(l)} \quad (A\backslash C_i)^l}{C_i^l}$), but as $i(l) = l$ we have $\frac{(C_i/A)^l \quad A^l}{C_i^l}$ (or $\frac{A^l \quad (A\backslash C_i)^l}{C_i^l}$) that is the same rule (written in a different way) that the algorithm of van Benthem uses for upward monotone functors, namely, $\frac{(\alpha^{+}_{l}, \beta) \quad \alpha_{l}}{\beta_{l}}$.

   (b) The functor is downward monotone. The algorithm of van Eijck uses the rule for the reverse map $\frac{(C_r/A)^l \quad A^{r(l)}}{C_r^l}$ (or $\frac{A^{r(l)} \quad (A\backslash C_r)^l}{C_r^l}$), let us remember that in the context of the algorithm of van Eijck $r(+) = -$, $r(-) = +$ and that in the context of the algorithm of van Benthem $-+ = -$ and $-- = +$, i.e., $r(l) = -l$; hence, the rule for the reverse map is the same rule (written in a different way) that the algorithm of van Benthem uses for downward monotone functors, namely, $\frac{(\alpha^{-}_{l}, \beta) \quad \alpha_{-l}}{\beta_{l}}$.

2. The functor is binary. The algorithm of van Benthem does not take into account a binary functor as the one in the algorithm of van Eijck $\frac{A^{f(l)} \quad (A\backslash C_f/B)^l \quad B^{f(l)}}{C_f^l}$. Nevertheless, if we do not take precedence between the contructors / and \, it is possible to get the same derivation only using unary constructors, in either of the two following ways.

$$\frac{\frac{A^{f(l)} \quad (A\backslash(C_f/B))^l}{(C_f/B)^l} \quad B^{f(l)}}{C_f^l}, \text{ or } \frac{A^{f(l)} \quad \frac{((A\backslash C_f)/B)^l \quad B^{f(l)}}{(A\backslash C_f)^l}}{C_f^l},$$

   achieving exactly the same effect, i.e., the binary functor inherits the polarity of the root and the arguments have polarity $f(l)$.

□

Chapter 4

Model-Theoretic Semantics for NL

Although NL was designed without a semantics in the style of model theory, in this chapter a semantics in that style is developed. The main reason for this is to define a model checking algorithm for NL. This will permit to know why a text does not imply a hypothesis, in the sense of section 2.1.

It should be remembered that a proof tree is needed to compute polarity. Hence, an automatic theorem prover for an extension to *AB* grammars is built. The algorithm of van Benthem is the one implemented to compute polarity.

4.1 Semantics for Natural Logic

We need to define what a model $\mathcal{M} = (D, [\![]\!])$ for NL is, where D is a non empty set called the *domain* of the model, and $[\![]\!]$ is the *meaning function* that maps expressions from a language to either elements or subsets of the domain.

We are going to extend a version of categorial grammars called *AB grammars* [MR12], this extension is in order to mark the monotonicity of functors. Types are constructed by the following rules

$$L ::= P|(L^s/L)|(L\backslash^s L),$$

where P is the set of primitive types (in our case e and t), and functors are constructed using the operators ${}^s/$ and $\backslash^s$, with $s \in \{+, -, \cdot\}$, these operators are intended to distinguish if the argument of a functor is either on the right or on the left, respectively. Also to mark whether a functor is upward monotone (+), downward monotone (−), or non-monotone ($\cdot$).

A model $\mathcal{M} = (D, [\![]\!])$ for NL has a family D of domains, beginning with the domain D_e corresponding to elements of type e (entity nouns), then the domain D_t corresponding to truth values, i.e., $D_t = \{0, 1\}$, and a family of domains $D_{X \overset{s}{\to} Y}$ corresponding to functors[1]

[1] In the case of $s = \cdot$, we are going to write $/, \backslash$, and $\to$; instead of ${}^{\cdot}/, \backslash^{\cdot}$, and $\overset{\cdot}{\to}$.

with domain X and range Y, and the meaning function $[\![]\!]$ is given through Definitions 4.1.1, 4.1.2, 4.1.3, and 4.1.9.

Definition 4.1.1. [Dow94] *Meaning of types.* Let $e, t \in P; X, Y \in L$, then

- $[\![e]\!] = D_e$,
- $[\![t]\!] = D_t$,
- $[\![(X^s/Y)]\!] = [\![(Y\backslash^s X)]\!] = D_{Y \overset{s}{\to} X}$ if $[\![Y]\!] = D_Y$, and $[\![X]\!] = D_X$. □

In general, if $X \in L$, then $[\![X]\!] = D_X$, if $x \in D_X$ is said that x has type X (in symbols, $x : X$).

Each word in a sentence has a type, and we want to give them a meaning as a sequence of types, as in the following definition.

Definition 4.1.2. *Juxtaposition of meanings.* If $X, Y \in L$, then $[\![X]\!][\![Y]\!] = [\![XY]\!]$. □

As it has been stated, to prove that a natural language expression is well formed (it is a sentence), a proof tree with root t has to be constructed, using the functional application rules for syntactic categories $\dfrac{X^s/Y \quad Y}{X}$, and $\dfrac{Y \quad Y\backslash^s X}{X}$; the proof tree in Figure 4.1 is an example.

$$\dfrac{\dfrac{\text{Dobie}}{(t^+/(t/e))} \qquad \dfrac{\dfrac{\text{brought}}{((t/e)/e)} \qquad \dfrac{\dfrac{\text{every}}{((((t/e)/e)\backslash^+(t/e))^-/(t/e))} \qquad \dfrac{\text{ball}}{(t/e)}}{(((t/e)/e)\backslash^+(t/e))}}{(t/e)}}{t}$$

Figure 4.1: Proof tree for the sentence *Dobie brought every ball*.

In our semantics, we have to do an equivalent analysis, i.e., we have to assign a meaning to functor application, therefore we have the following definition.

Definition 4.1.3. *Meaning of function application.* Let $X, Y \in L$, and let Z_1, and Z_2 be sequences of types (possibly empty), then

$$[\![Z_1\ (X^s/Y)\ Y\ Z_2]\!] = [\![Z_1\ Y\ (Y\backslash^s X)\ Z_2]\!] = [\![Z_1\ X\ Z_2]\!].$$

□

Example 4.1.4. Taking the types of the words of the sentence *Dobie brought every ball*, and computing their meaning by Definitiions 4.1.1 and 4.1.3, we get the domain D_t.

$$\overbrace{[\![(t^+/(t/e))}^{Z_1}\ \overbrace{((t/e)/e)}^{(X^s/Y)}\ \overbrace{((((t/e)/e)\backslash^+(t/e))^-/(t/e))}^{Y}\ \overbrace{(t/e)]\!]}^{X} \overset{4.1.3}{=}$$
$$\overbrace{[\![(t^+/(t/e))}^{Z_1}\ \overbrace{((t/e)/e)}^{Y}\ \overbrace{(((t/e)/e)\backslash^+(t/e))}^{(Y\backslash^s X)}]\!] \overset{4.1.3}{=}$$
$$\overbrace{[\![(t^+/(t/e))}^{Z_1}\ \overbrace{(t/e)]\!]}^{X} \overset{4.1.3}{=}$$
$$\overbrace{[\![(t^+/(t/e))}^{(X^s/Y)}\ \overbrace{(t/e)]\!]}^{Y} =$$
$$\overbrace{[\![t]\!]}^{X} \overset{4.1.1}{=} D_t.$$

□

Now, we prove that if a proof tree with root $Z \in L$ is constructed, then Z has meaning D_Z.

Theorem 4.1.5. *If $Z \in L$ is the root of a proof tree, then $[\![Z]\!] = D_Z$.*

Proof. The proof is by induction on the depth of the proof tree. If the depth of the proof tree is 0, we have the following cases:

1. $Z = e$, therefore $[\![e]\!] \overset{4.1.1}{=} D_e$,
2. $Z = t$, therefore $[\![t]\!] \overset{4.1.1}{=} D_t$,
3. $Z = (X^s/Y)$, therefore $[\![(X^s/Y)]\!] \overset{4.1.1}{=} D_{Y \to^s X}$ if $[\![Y]\!] = D_Y$, and $[\![X]\!] = D_X$,
4. $Z = (Y\backslash^s X)$, similar to $Z = (X^s/Y)$.

If the depth of the proof tree is n, we have the following cases:

1. $Z = e$, therefore it is the root of one of the two following trees:
   (a) $\dfrac{\overset{\vdots}{(e^s/Y)} \quad \overset{\vdots}{Y}}{e}$, then (e^s/Y), and Y are the roots of proof trees with depth less than n, hence by induction hypothesis we have $[\![(e^s/Y)]\!]\ [\![Y]\!] \overset{4.1.2}{=} [\![(e^s/Y)\ Y]\!] \overset{4.1.3}{=} [\![e]\!] \overset{4.1.1}{=} D_e$,

(b) $\dfrac{\vdots\ Y \quad (Y\backslash^s e)}{e}$, then Y, and $(Y\backslash^s e)$ are the roots of proof trees with depth less than n, hence by induction hypothesis we have $[\![Y]\!]\ [\![(Y\backslash^s e)]\!] \overset{4.1.2}{=} [\![Y\ (Y\backslash^s e)]\!] \overset{4.1.3}{=} [\![e]\!] \overset{4.1.1}{=} D_e$.

2. $Z = t$, similar to $Z = e$.

3. $Z = (Z_1^s/Z_2)$, therefore it is the root of one of the two following trees:

(a) $\dfrac{((Z_1^s/Z_2)^{s'}/Z_3) \quad \vdots\ Z_3}{(Z_1^s/Z_2)}$, then $((Z_1^s/Z_2)^{s'}/Z_3)$, and Z_3 are the roots of proof trees with depth less than n, hence by induction hypothesis we have $[\![((Z_1^s/Z_2)^{s'}/Z_3)]\!]\ [\![Z_3]\!] \overset{4.1.2}{=} [\![((Z_1^s/Z_2)^{s'}/Z_3)\ Z_3]\!] \overset{4.1.3}{=} [\![(Z_1^s/Z_2)]\!] \overset{4.1.1}{=} D_{Z_2 \overset{s}{\to} Z_1}$,

(b) $\dfrac{\vdots\ Z_3 \quad (Z_3\backslash^{s'}(\vdots\ Z_1^s/Z_2))}{(Z_1^s/Z_2)}$, then Z_3, and $(Z_3\backslash^{s'}(Z_1^s/Z_2))$ are the roots of proof trees with depth less than n, hence by induction hypothesis we have $[\![Z_3]\!]\ [\![(Z_3\backslash^{s'}(Z_1^s/Z_2))]\!] \overset{4.1.2}{=} [\![Z_3\ (Z_3\backslash^{s'}(Z_1^s/Z_2))]\!] \overset{4.1.3}{=} [\![(Z_1^s/Z_2)]\!] \overset{4.1.1}{=} D_{Z_2 \overset{s}{\to} Z_1}$.

4. $Z = (Z_1\backslash^s Z_2)$, similar to $Z = (Z_1^s/Z_2)$.

□

The next theorem proves that if $Z \in L$ has meaning D_Z, then there exists a proof tree whose root is Z.

Theorem 4.1.6. *Let $Z \in L$, if $[\![Z]\!] = D_Z$, then there exists a proof tree whose root is Z.*

Proof. The proof is by induction on the number of times that Definition 4.1.3 was used to get D_Z.

1. If Definition 4.1.3 was used 0 times, we have four cases:

(a) If $[\![e]\!] \overset{4.1.1}{=} D_e$, then we construct the tree $\dfrac{}{e}$, i.e., a tree with depth 0, and root e;

(b) If $[\![t]\!] \overset{4.1.1}{=} D_t$, similar to $[\![e]\!] \overset{4.1.1}{=} D_e$;

(c) If $[\![(X^s/Y)]\!] \overset{4.1.1}{=} D_{Y \overset{s}{\to} X}$, then we construct the tree $\dfrac{}{(X^s/Y)}$, i.e., a tree with depth 0, and root (X^s/Y);

(d) If $[\![(Y\backslash^s X)]\!] \overset{4.1.1}{=} D_{Y \overset{s}{\to} X}$, similar to $[\![(X^s/Y)]\!] \overset{4.1.1}{=} D_{Y \overset{s}{\to} X}$.

2. If Definition 4.1.3 was used n times, we have eight cases:

(a) $D_e \overset{4.1.1}{=} [\![e]\!] \overset{4.1.3}{=} [\![(e^s/Y)\ Y]\!] \overset{4.1.2}{=} [\![(e^s/Y)]\!]\ [\![Y]\!]$, but to get $[\![(e^s/Y)]\!]$, and $[\![Y]\!]$ Definition 4.1.3 was used less than n times, hence by induction hypothesis there exist proof trees which are root, namely $\dfrac{\vdots\ (e^s/Y) \quad \vdots\ Y}{}$. Now, using the functional application rule for syntactic categories, we get the tree $\dfrac{\vdots\ (e^s/Y) \quad \vdots\ Y}{e}$;

(b) $D_e \overset{4.1.1}{=} [\![e]\!] \overset{4.1.3}{=} [\![Y\ (Y\backslash^s e)]\!] \overset{4.1.2}{=} [\![Y]\!]\ [\![(Y\backslash^s e)]\!]$, but to get $[\![Y]\!]$, and $[\![(Y\backslash^s e)]\!]$ Definition 4.1.3 was used less than n times, hence by induction hypothesis there exist proof trees which are root, namely $\dfrac{\vdots\ Y \quad \vdots\ (Y\backslash^s e)}{}$. Now, using the functional application rule for syntactic categories, we get the tree $\dfrac{\vdots\ Y \quad \vdots\ (Y\backslash^s e)}{e}$;

(c) $D_t \overset{4.1.1}{=} [\![t]\!] \overset{4.1.3}{=} [\![(t^s/Y)\ Y]\!] \overset{4.1.2}{=} [\![(t^s/Y)]\!]\ [\![Y]\!]$, similar to $D_e \overset{4.1.1}{=} [\![e]\!] \overset{4.1.3}{=} [\![(e^s/Y)\ Y]\!]$;

(d) $D_t \overset{4.1.1}{=} [\![t]\!] \overset{4.1.3}{=} [\![Y\ (Y\backslash^s t)]\!]$, similar to $D_e \overset{4.1.1}{=} [\![e]\!] \overset{4.1.3}{=} [\![Y\ (Y\backslash^s e)]\!]$;

(e) $D_{Z_2 \overset{s}{\to} Z_1} \overset{4.1.1}{=} [\![(Z_1^s/Z_2)]\!] \overset{4.1.3}{=} [\![((Z_1^s/Z_2)^{s'}/Z_3)\ Z_3]\!] \overset{4.1.2}{=} [\![((Z_1^s/Z_2)^{s'}/Z_3)]\!]\ [\![Z_3]\!]$, but to get $[\![((Z_1^s/Z_2)^{s'}/Z_3)]\!]$, and $[\![Z_3]\!]$ Definition 4.1.3 was used less than n times, hence by induction hypothesis there exist proof trees which are root, namely $\dfrac{\vdots\ ((Z_1^s/Z_2)^{s'}/Z_3) \quad \vdots\ Z_3}{}$. Now, using the functional application rule for syntactic categories, we get the tree $\dfrac{\vdots\ ((Z_1^s/Z_2)^{s'}/Z_3) \quad \vdots\ Z_3}{(Z_1^s/Z_2)}$;

(f) $D_{Z_2 \overset{s}{\to} Z_1} \overset{4.1.1}{=} [\![(Z_1^s/Z_2)]\!] \overset{4.1.3}{=} [\![Z_3\ (Z_3\backslash^{s'}(Z_1^s/Z_2))]\!] \overset{4.1.2}{=} [\![Z_3]\!]\ [\![(Z_3\backslash^{s'}(Z_1^s/Z_2))]\!]$, but to get $[\![Z_3]\!]$, and $[\![(Z_3\backslash^{s'}(Z_1^s/Z_2))]\!]$ Definition 4.1.3 was used less than n times, hence by induction hypothesis there exist proof trees which are root, namely $\dfrac{\vdots\ Z_3 \quad \vdots\ (Z_3\backslash^{s'}(Z_1^s/Z_2))}{}$. Now, using the functional application rule for syntactic categories, we get the tree $\dfrac{\vdots\ Z_3 \quad \vdots\ (Z_3\backslash^{s'}(Z_1^s/Z_2))}{(Z_1^s/Z_2)}$;

(g) $D_{Z_2 \overset{s}{\to} Z_1} \overset{4.1.1}{=} [\![(Z_1\backslash^s Z_2)]\!] \overset{4.1.3}{=} [\![((Z_1\backslash^s Z_2)^{s'}/Z_3)\ Z_3]\!]$, similar to $D_{Z_2 \overset{s}{\to} Z_1} \overset{4.1.1}{=} [\![(Z_1^s/Z_2)]\!] \overset{4.1.3}{=} [\![((Z_1^s/Z_2)^{s'}/Z_3)\ Z_3]\!]$;

(h) $D_{Z_2 \overset{s}{\to} Z_1} \overset{4.1.1}{=} [\![(Z_1\backslash^s Z_2)]\!] \overset{4.1.3}{=} [\![Z_3\ (Z_3\backslash^{s'}(Z_1\backslash^s Z_2))]\!]$, similar to $D_{Z_2 \overset{s}{\to} Z_1} \overset{4.1.1}{=} [\![(Z_1^s/Z_2)]\!] \overset{4.1.3}{=} [\![Z_3\ (Z_3\backslash^{s'}(Z_1^s/Z_2))]\!]$.

□

With theorems 4.1.5 and 4.1.6 it is proved that the proposed semantics is sound and complete with respect to the proof system of the *AB* grammars.

To know when a sentence N' is inferred from a sentence N, first we have to define what a subexpression of a natural language expression is.

Definition 4.1.7. *Subexpression.* Let $N = w_1 w_2 \cdots w_n, n \geq 1$, be a natural language expression, where each word $w_i \in D_{\alpha_i}$, it is said that M is a *subexpression* of N if and only if one of the following holds:

1. $M = w_i, 1 \leq i \leq n$;
2. $M = w_i M', 1 \leq i \leq n-1$, where M' is a subexpression of N, and ($w_i \in D_{\alpha\to\beta}$ and $M' \in D_\alpha$) or ($w_i \in D_\alpha$ and $M' \in D_{\alpha\to\beta}$) holds;
3. $M = M' w_i, 2 \leq i \leq n$, where M' is a subexpression of N, and ($w_i \in D_{\alpha\to\beta}$ and $M' \in D_\alpha$) or ($w_i \in D_\alpha$ and $M' \in D_{\alpha\to\beta}$) holds;
4. $M = M'M''$, where M' and M'' are subexpressions of N, and ($M' \in D_{\alpha\to\beta}$ and $M'' \in D_\alpha$) or ($M' \in D_\alpha$ and $M'' \in D_{\alpha\to\beta}$) holds. □

Example 4.1.8. Let N = *Dobie brought every ball*, according to clause 1 of Definition 4.1.7, we have that *Dobie, brought, every*, and *ball* are subexpressions of N; looking at Figure 4.1 we have that *every ball* is also a subexpression by clause 2 of Definition 4.1.7, because *every:* $((((t/e)/e)\backslash^+(t/e))^-/(t/e))$, and *ball:* (t/e), by the same clause are also subexpressions *brought every ball*, and *Dobie brought every ball*. □

The following definition is important because it states what the meaning of a natural language expression is.

Definition 4.1.9. *Meaning of expressions.* Let $x : X$, be a natural language expression, we define the *meaning of the expression* x as the set $[\![x]\!]_X = \{y \in D_X \mid y \leq x\}$. □

Now, we can define when a natural language expression entails another one, the idea behind is to connect the notions of polarity, subexpression and meaning of an expression. Also this definition is the kernel of our method of Model Checking for NL.

Definition 4.1.10. *Entailment on the same subexpression.* Let $N(M)$, and $N(M')$ be two natural language expressions with $M, M' : \alpha, M \neq M'$, we define that $N(M)$ *entails on the same subexpression* $N(M')$ (symbolically $N(M) \overset{*}{\models} N(M')$) in the following way:

$$N(M) \overset{*}{\models} N(M') \text{ if and only if } \begin{cases} M \text{ has positive polarity and } [\![M]\!]_\alpha \subseteq [\![M']\!]_\alpha, \text{ or} \\ M \text{ has negative polarity and } [\![M']\!]_\alpha \subseteq [\![M]\!]_\alpha. \end{cases}$$

□

According to Israel [Isr11], Ladusaw found that most of Negative Polarity Items in English are licenced by *downward entailing propositional operators*, these operators reverse the relation of semantic strength among expressions. Similarly, Positive Polarity Items are licenced by *upward entailing propositional operators*, they preserve the relation of semantic strength among expressions.

Considering that an expression e_1 is semantically stronger than the expression e_2, if the set denoted by the meaning of e_1 is a subset of the set denoted by the meaning of e_2, then the Definition 4.1.10 meets Ladusaw's findings.

Example 4.1.11. A proof tree with polarity marks for the natural language expression *An elk ran* is shown in Figure 4.5.

$$\dfrac{\dfrac{\dfrac{\text{An}}{((t^{+}/(t/e))^{+}/(t/e))} \quad \dfrac{\text{elk}}{(t/e)}}{(t^{+}/(t/e))} \quad \dfrac{\text{ran}}{(t/e)}}{t}$$

Figure 4.2: Proof tree with polarity marks for the natural language expression *An elk ran*.

If N = *An elk ran*, M = *elk* y M′ = *mammal*, then we can say that *An elk ran* $\overset{*}{\models}$ *A mammal ran*, because *elk* has positive polarity, and $[\![elk]\!]_{(t/e)} \subseteq [\![mammal]\!]_{(t/e)}$. □

Example 4.1.12. If N = *An elk ran*, M = *elk* and M′ = *animal* we can say that *An elk ran* $\overset{*}{\models}$ *An animal ran*, because *elk* has positive polarity and $[\![elk]\!]_{(t/e)} \subseteq [\![animal]\!]_{(t/e)}$. □

Theorem 4.1.13 proves that our notion of entailment on the same subexpression is sound with respect to the notion of implication in the proof theory of NL.

Theorem 4.1.13. *If* $N(M) \vdash N(M')$, *then* $N(M) \overset{*}{\models} N(M')$.

Proof. We have two cases:

1. M has positive polarity, then $M \leq_\alpha M'$, hence $\{y \in D_\alpha \mid y \leq M\} \subseteq \{y \in D_\alpha \mid y \leq M'\}$, which implies that $[\![M]\!]_\alpha \subseteq [\![M']\!]_\alpha$, therefore $N(M) \overset{*}{\models} N(M')$.

2. M has negative polarity, then $M' \leq_\alpha M$, hence $\{y \in D_\alpha \mid y \leq M'\} \subseteq \{y \in D_\alpha \mid y \leq M\}$, which implies that $[\![M']\!]_\alpha \subseteq [\![M]\!]_\alpha$, therefore $N(M) \overset{*}{\models} N(M')$.

□

Now, in Theorem 4.1.14 it is proved that the concept of entailment on the same subexpression is complete with respect to the proof theory of NL.

Theorem 4.1.14. *If* $N(M) \overset{*}{\vDash} N(M')$, *then* $N(M) \vdash N(M')$.

Proof. We have two cases:

1. M has positive polarity, then $[\![M]\!]_\alpha \subseteq [\![M']\!]_\alpha$, hence $[\![M]\!]_\alpha = \{y \in D_\alpha \mid y \leq M\} \subseteq \{y \in D_\alpha \mid y \leq M'\} = [\![M']\!]_\alpha$, as D_α is partially ordered, it is implied that $M \leq_\alpha M'$, therefore $N(M) \vdash N(M')$.

2. M has negative polarity, then $[\![M']\!]_\alpha \subseteq [\![M]\!]_\alpha$, hence $[\![M']\!]_\alpha = \{y \in D_\alpha \mid y \leq M'\} \subseteq \{y \in D_\alpha \mid y \leq M\} = [\![M]\!]_\alpha$, as D_α is partially ordered, it is implied that $M' \leq_\alpha M$, therefore $N(M) \vdash N(M')$.

□

With Theorems 4.1.13 and 4.1.14 is warranted that the method of Model Checking for NL, presented in 4.3 is correct with respect to the Proof Theory of NL.

To define a method of Model Checking for Natural Logic is needed to chain entailments on more than one subexpression, this is done in the following way:

Definition 4.1.15. *Entailment.* Let N, and N′ be two natural language expressions with $N \neq N'$, we define that N *entails* N′ (symbolically $N \vDash N'$) as follows:

$$N \vDash N' \text{ if and only if } \begin{cases} N' = N(M') \text{ and } N(M) \overset{*}{\vDash} N(M'), \text{ or} \\ N'' = N(M''), N(M) \overset{*}{\vDash} N(M'') \text{ and } N'' \vDash N'. \end{cases}$$

□

4.2 Proof Theory for an Extension of *AB* Grammars

Before the method of model checking can be constructed, it is necessary to build proof trees for English sentences using the function application rules $\dfrac{X^s/Y \quad Y}{X}$, and $\dfrac{Y \quad Y\backslash^s X}{X}$. Therefore, an automatic theorem prover for the extension of *AB* grammars is developed in this section.

A natural language expression $nlexp$ is represented by the list $[w_1 : X_1, \ldots, w_n : X_n]$, where w_i is an english word in $nlexp$, and X_i is its type, $1 \leq i \leq n$. As an example, the natural language expression *Dobie didn't bring every ball* is represented by the list

$$[Dobie : (t^+/(t/e)), didn't : ((t/e)^-/(t/e)), bring : ((t/e)/e),$$
$$every : ((((t/e)/e)\backslash^+(t/e))^-/(t/e)), ball : (t/e)].$$

$$\dfrac{\dfrac{\text{Dobie}}{(t^{+}/(t/e))} \quad \dfrac{\dfrac{\text{didn't}}{((t/e)^{-}/(t/e))} \quad \dfrac{\dfrac{\text{bring}}{((t/e)/e)} \quad \dfrac{\dfrac{\text{every}}{((((t/e)/e)\backslash^{+}(t/e))^{-}/(t/e))} \quad \dfrac{\text{ball}}{(t/e)}}{(((t/e)/e)\backslash^{+}(t/e))}}{(t/e)}}{(t/e)}}{t}$$

Figure 4.3: Proof tree for the sentence *Dobie didn't bring every ball*, in an extension of *AB* grammars.

As it is known, there are natural language expressions that admit more than one syntactic analysis. Evenmore, sometimes it is possible to use the inference rules in more than one pair of adjacent words.

Hence, we need to look for those pairs of words that may conform the initial subtrees of possible proof trees. Also, for each initial subtree we need to know the list of pairs $w : X$ on the left, and on the right of the subtree, these three elements will be called an environment.

For example, for the proof tree in Figure 4.3 its unique initial subtree is

$$\dfrac{\dfrac{\textit{every}}{((((t/e)/e)\backslash^{+}(t/e))^{-}/(t/e))} \quad \dfrac{\texttt{ball}}{(t/e)}}{(((t/e)/e)\backslash^{+}(t/e))},$$

and its lists of pairs are $[\texttt{Dobie} : (t^{+}/(t/e)), \texttt{didn't} : ((t/e)^{-}/(t/e)), \texttt{bring} : ((t/e)/e)]$ on the left, and $[]$ on the right; the algorithm BFSTL returns a list of environments of the form

$$[([w_1 : X_1, \ldots, w_{i_1-1} : X_{i_1-1}], \dfrac{\dfrac{w_{i_1}}{X_{i_1}} \quad \dfrac{w_{i_1+1}}{X_{i_1+1}}}{X_1}, [w_{i_1+2} : X_{i_1+2}, \ldots, w_n : X_n]), \ldots,$$

$$([w_1 : X_1, \ldots, w_{i_m-1} : X_{i_m-1}], \dfrac{\dfrac{w_{i_m}}{X_{i_m}} \quad \dfrac{w_{i_m+1}}{X_{i_m+1}}}{X_m}, [w_{i_m+2} : X_{i_m+2}, \ldots, w_n : X_n])],$$

$$1 \leq i_1, i_m, m < n.$$

```
FUNCTION BFSTL(left, current, envs)
 1  switch
 2    case current = [] and envs = [] :
 3         raise exception NoFirstSubTree
 4    case current = [] :
 5         return envs
 6    case current = [w_n : X_n] and envs = [] :
 7         raise exception NoFirstSubTree
 8    case current = [w_n : X_n] :
 9         return envs
10    case current = [w_i : X_i, w_{i+1} : X_{i+1}, w_{i+2} : X_{i+2}, ..., w_n : X_n] :
11         if X_i = X^s/X_{i+1} or X_{i+1} = X_i\^sX
12           then BFSTL(left ⊎ [w_i : X_i], [w_{i+1} : X_{i+1}, ..., w_n : X_n],
                                                  w_i        w_{i+1}
13                    envs ⊎ [(left, ------------------------------,
                                          X_i          X_{i+1}
14                    [w_{i+2} : X_{i+2}, ..., w_n : X_n])]
15           else BFSTL(left ⊎ [w_i : X_i], [w_{i+1} : X_{i+1}, ..., w_n : X_n],
16                    envs)
17  end
```

The algorithm BFSTL has three input parameters: left the pairs $w_j : X_j, 1 \leq j < i$ already processed; current the pairs $w_k : X_k, i \leq k \leq n$ which are not being processed; and envs the list of environments found until now.

In general (line 10), if some inference rule can be applied to X_i and X_{i+1} (line 11), then a recursive call is performed appending ($\uplus$) left and the list $[w_i : X_i]$; stating that $[w_{i+1} : X_{i+1}, \ldots, w_n : X_n]$ is the new current, and appending envs and the list with the environment found $[(\text{left}, \frac{\frac{w_i}{X_i} \quad \frac{w_{i+1}}{X_{i+1}}}{X}, [w_{i+2} : X_{i+2}, \ldots, w_n : X_n])]$ (lines 12-14).

If no rule can be applied (lines 15 and 16), a recursive call is made indicating that a word has been processed, and leaving envs without change.

If no environment was found (lines 2 and 6) then the NoFirstSubTree exception is raised (lines 3 and 7). If there are no more elements to process (line 4) or there is only one element (line 8), then the work has been done and the list of environments envs is returned (lines 5 and 9).

The algorithm BAWPT builds a proof tree from a list of environments. If the list of environments is not empty (lines 4 and 5), then the algorithm BAPT is called with the

elements of the first environment in the list, and the type of the root of the first subtree (line 6).

Function bawpt($environmentsList$)

```
1  switch
2    case environmentsList = [] :
3         raise exception NoProofTree
4    case environmentsList =
5         [(left_1, fst_1, right_1), ..., (left_n, fst_n, right_n)] :
6         tree ← bapt(left_1, fst_1, right_1, extracType(fst_1))
7         if tree = S/X
8           then bawpt([(left_2, fst_2, right_2), ...,
9                 (left_n, fst_n, right_n)])
10          else return tree
11 end
```

If the algorithm bapt could not build a proof tree, then a recursive call is performed with the rest of the list of environments (lines 8 and 9). If the algorithm bapt built a proof tree, this is returned (line 10). If the environment list is empty, then it was not possible to build a proof tree and the exception NoProofTree is raised (lines 2 and 3).

The algorithm extracType merely returns the root of a unary tree (lines 2 and 3), or the root of a binary tree (lines 4 and 5). This is used in line 6 of the algorithm bawpt.

Function extracType($tree$)

```
1  switch
2    case tree = S/X :
3         return X
4    case tree = proofSubTree_l  proofSubTree_r / X :
5         return X
```

The purpose of the algorithm bapt is to build a proof tree. It takes four arguments: the list left of pairs $w : X$ on the left of the proof subtree proofSubTree, the proof subtree proofSubTree already built, the list right of pairs $w : X$ on the right of the proof subtree proofSubTree, and the type X of the root of the proof subtree proofSubTree.

If left and right are empty, then a proof tree has been constructed, and there is nothing more to process (lines 2 and 3).

If left is not empty, but right is empty, then to construct a new proof subtree it is needed that the root X of subProofTree can combine with the type X_i of the last element

of left, this is possible when $X = X_i\backslash^s X'$ is the functor and X_i is the argument, or when X is the argument and $X_i = X'^s/X$ is the functor, if that is the case, then a recursive call is performed pointing out that: $w_i : X_i$ has been processed, the new proof subtree $\dfrac{\dfrac{w_i}{X_i} \quad \mathtt{proofSubTree}}{X'}$ has been constructed, right is still empty, the root of the new proof subtree is X' (lines 4-7).

FUNCTION BAPT(left, proofSubTree, right, X)

1 **switch**
2   **case** left = [] and right = [] :
3   **return** proofSubTree
4   **case** left = $[w_1 : X_1, \ldots, w_i : X_i]$ and right = [] :
5   **if** $X = X_i\backslash^s X'$ or $X_i = X'^s/X$
6   **then** BAPT($[w_1 : X_1, \ldots, w_{i-1} : X_{i-1}]$,
7         $\dfrac{\dfrac{w_i}{X_i} \quad \mathtt{proofSubTree}}{X'}$, [], X')
8   **else return** $\dfrac{''''}{e}$
9   **case** left = [] and right = $[w_j : X_j, \ldots, w_n : X_n]$:
10   **if** $X = X'^s/X_j$ or $X_j = X\backslash^s X'$
11   **then** BAPT([], $\dfrac{\mathtt{proofSubTree} \quad \dfrac{w_j}{X_j}}{X'}$,
12         $[w_{j+1} : X_{j+1}, \ldots, w_n : X_n]$, X')
13   **else return** $\dfrac{''''}{e}$
14   **case** left = $[w_1 : X_1, \ldots, w_i : X_i]$ and
15   right = $[w_j : X_j, \ldots, w_n : X_n]$:
16   **switch**
17   **case** $X = X_i\backslash^s X'$ or $X_i = X'^s/X$:
18   BAPT($[w_1 : X_1, \ldots, w_{i-1} : X_{i-1}]$,
19         $\dfrac{\dfrac{w_i}{X_i} \quad \mathtt{proofSubTree}}{X'}$,
20         $[w_j : X_j, \ldots, w_n : X_n]$, X')
21   **case** $X = X'^s/X_j$ or $X_j = X\backslash^s X'$:
22   BAPT($[w_1 : X_1, \ldots, w_i : X_i]$,
23         $\dfrac{\mathtt{proofSubTree} \quad \dfrac{w_j}{X_j}}{X'}$,
24         $[w_{j+1} : X_{j+1}, \ldots, w_n : X_n]$, X')
25   **case default** :
26   **return** $\dfrac{''''}{e}$
27 **end**

If it was not possible to build a new proof subtree, then the tree $\frac{''''}{e}$ is returned (lines 8, 13 and 23), so that the algorithm BAWPT tries to build a proof tree with the remaining environments.

If left is empty, but right is not empty, then to construct a new proof subtree it is needed that the root X of subProofTree can combine with the type X_j of the first element of right, this is possible when $X = X'^s/X_j$ is the functor and X_j is the argument, or when X is the argument and $X_j = X\backslash^s X'$ is the functor, if that is the case, then a recursive call is performed pointing out that: left is still empty, the new proof subtree $\frac{\text{proofSubTree} \quad \frac{w_j}{X_j}}{X'}$ has been constructed, $w_j : X_j$ has been processed, the root of the new proof subtree is X' (lines 9-12).

If left and right are not empty, then the root X of proofSubTree can combine with the type X_i of the last element of left (lines 17-20); or the root X of proofSubTree can combine with the type X_j of the first element of left (lines 21-24), just like the respective previous cases.

The algorithm BUILDPROOFTREE passes the appropriate initial values to BFSTL in order to get a list of environments, this list is passed to BAWPT, which constructs a proof tree.

FUNCTION BUILDPROOFTREE(nlexp)

1 **return** BAWPT(BFSTL([], nlexp, []))

2 **end**

To compute polarity in our extension to *AB* grammars, the algorithm of van Benthem is adapted as follows.

Algorithm 4.2.1. van Benthem's polarity algorithm adapted to an extension of *AB* grammars.

1. Label the root with +.

2. Propagate notations up the tree.

   (a) If a node of type X is labeled l and its children are of type (X^s/Y) and Y, then the former child is to be labeled l and the latter child is to be labeled $l \circ s$ (diagrammatically, $\frac{\overset{(X^s/Y)}{l} \quad \overset{Y}{l\circ s}}{\underset{l}{X}}$).

   (b) If a node of type X is labeled l and its children are of type Y and $(Y\backslash^s X)$, then the former child is to be labeled $l \circ s$ and the latter child is to be labeled l (diagrammatically, $\frac{\overset{Y}{l\circ s} \quad \overset{(Y\backslash^s X)}{l}}{\underset{l}{X}}$).

As an example we have the proof tree of Figure 4.4.

$$\dfrac{\dfrac{\text{Dobie}}{(t^{+}/(t/e))}_{+} \quad \dfrac{\dfrac{\text{didn't}}{((t/e)^{-}/(t/e))}_{+} \quad \dfrac{\dfrac{\text{bring}}{((t/e)/e)}_{-} \quad \dfrac{\dfrac{\text{every}}{((((t/e)/e)\backslash^{+}(t/e))^{-}/(t/e))}_{-} \quad \dfrac{\text{ball}}{(t/e)}_{+}}{(((t/e)/e)\backslash^{+}(t/e))}_{-}}{(t/e)}_{-}}{(t/e)}_{+}}{t}_{+}$$

Figure 4.4: Proof tree with polarity marks for the sentence *Dobie didn't bring every ball*, using the adapted algorithm of van Benthem, in an extension of *AB* grammars.

The algorithm POLALG encodes the algorithm of van Benthem more precisely. It returns a proof tree with polarity marks. Its arguments are: the polarity label l for the root of $tree$, and the proof tree $tree$.

FUNCTION POLALG(l, $tree$)

1 **switch**

2 **case** $tree = \dfrac{w}{X}$:

3 **return** $\dfrac{w}{X}_{l}$

4 **case** $tree = \dfrac{\dfrac{\vdots}{(X^{s}/Y)} \quad \dfrac{\vdots}{Y}}{X}$:

5 $lptp \leftarrow$ POLALG$(l, \dfrac{\vdots}{(X^{s}/Y)})$

6 $rptp \leftarrow$ POLALG$(l \circ s, \dfrac{\vdots}{Y})$

7 **return** $\dfrac{lptp \quad rptp}{X}_{l}$

8 **case** $tree = \dfrac{\dfrac{\vdots}{Y} \quad \dfrac{\vdots}{(Y\backslash^{s}X)}}{X}$:

9 $lptp \leftarrow$ POLALG$(l \circ s, \dfrac{\vdots}{Y})$

10 $rptp \leftarrow$ POLALG$(l, \dfrac{\vdots}{(Y\backslash^{s}X)})$

11 **return** $\dfrac{lptp \quad rptp}{X}_{l}$

12 **end**

If the current tree is a unary tree, then it returns a unary tree marking the root with l

(lines 2 and 3).

If the current tree is a binary tree and the functor is the left subtree, then it recursively propagates the polarity $\mathtt{l}$ on the left subtree lptp, and also it recursively propagates the polarity $\mathtt{l} \circ \mathtt{s}$ on the right subtree rptp. Finally it returns a binary tree marking the root with $\mathtt{l}$, and having lptp and rptp as left and right subtrees, respectively (lines 4-7).

If the current tree is a binary tree and the functor is the right subtree, then it recursively propagates the polarity $\mathtt{l} \circ \mathtt{s}$ on the left subtree lptp, and also it recursively propagates the polarity $\mathtt{l}$ on the right subtree rptp. Finally it returns a binary tree marking the root with $\mathtt{l}$, and having lptp and rptp as left and right subtrees, respectively (lines 8-11).

The algorithm POLARITY returns a proof tree with polarity marks, it receives as argument the representation of a natural language expression nlexp, as it was discussed in section 4.2. POLARITY calls POLALG with the mark of positive polarity and the proof tree for nlexp.

FUNCTION POLARITY(nlexp)

1 **return** POLALG(+, BUILDPROOFTREE(nlexp))

2 **end**

Example 4.2.2. A proof tree with polarity marks for the natural language expression *An elk ran* is shown in Figure 4.5.

$$\dfrac{\dfrac{\dfrac{\mathtt{An}}{((\mathtt{t}^{+}/(\mathtt{t}/\mathtt{e}))^{+}/(\mathtt{t}/\mathtt{e}))}_{+} \quad \dfrac{\mathtt{elk}}{(\mathtt{t}/\mathtt{e})}_{+}}{(\mathtt{t}^{+}/(\mathtt{t}/\mathtt{e}))}_{+} \quad \dfrac{\mathtt{ran}}{(\mathtt{t}/\mathtt{e})}_{+}}{\mathtt{t}}_{+}$$

Figure 4.5: Proof tree with polarity marks for the natural language expression *An elk ran*, in an extension of *AB* grammars.

If N = *An elk ran*, M = *elk* y M′ = *mammal*, then we can say that *An elk ran* $\overset{*}{\models}$ *A mammal ran*, because *elk* has positive polarity, and $[\![elk]\!]_{(\mathtt{t}/\mathtt{e})} \subseteq [\![mammal]\!]_{(\mathtt{t}/\mathtt{e})}$. □

Example 4.2.3. If N = *An elk ran*, M = *elk* and M′ = *animal* we can say that *An elk ran* $\overset{*}{\models}$ *An animal ran*, because *elk* has positive polarity and $[\![elk]\!]_{(\mathtt{t}/\mathtt{e})} \subseteq [\![animal]\!]_{(\mathtt{t}/\mathtt{e})}$. □

4.3 Model Checking for Natural Logic

Now, we define the algorithms ModChkNatLog, EntailsAll, and EntailsOne.

```
FUNCTION ModChkNatLog(N, N′)
1  Form the set DifSub with all the pairs M, M′ subexpressions of N, N′, such that M ≠ M′
2  return EntailsAll(DifSub, true)
3  end
```

The algorithm ModChkNatLog has as input the natural language expressions N, and N′, it returns the result of the algorithm EntailsAll. The purpose of ModChkNatLog is to warrant that EntailsAll receives, in the set DifSub, all the pairs (M, M'), where M, M' are respectively subexpressions of N, N' that make $N \neq N'$. Also it receives true the first time that it is called.

```
EntailsAll(DifSub, flag)
1  if DifSub = ∅
2    then return flag
3    else  From DifSub take a pair (M, M′)
4          if type(M) = type(M′)
5            then if not EntailsOne(M, M′)
6                   then if polarity(M) = polarity(M′)
7                          then switch
8                                 case polarity(M) = + :
9                                      write: ⟦M⟧α ⊈ ⟦M′⟧α
10                                case polarity(M) = − :
11                                     write: ⟦M′⟧α ⊈ ⟦M⟧α
12                                case default :
13                                     write: There is not a relationship between M and M′.
14                         else write: There is a possible contradiction between M and M′.
15                        EntailsAll(DifSub − (M, M′), false ∧ flag)
16                  else EntailsAll(DifSub − (M, M′), true ∧ flag)
17           else write: The syntactic structures of M and M′ are not equal.
18                return false
19  end
```

EntailsAll tries to find models that form counterexamples to the entailment of two natural language expressions. The algorithm EntailsAll codes Definition 4.1.15, it takes two parameters as input: DifSub containing the pairs of subexpressions (M, M') that make N and N′ different, and the variable flag, which records (line 15) if a counterexample, that is falsifying the entailment, has been found.

As it is implicit in Definition 4.1.10, a natural language expression $N(M)$ entails $N(M')$ if they vary in subexpressions M and M' of the same type, and the meaning of M contains (is contained in) the meaning of M' according to their polarity.

Hence, the main purpose of EntailsAll is to process a pair (M, M') from DifSub with the same type (lines 3 and 4), then if EntailsOne fails (line 5) and M has the same polarity as M' (line 6), it means that a counterexample has been found, and it writes the

cause of the failure (lines 9, 11, 13, 14), then it calls itself removing the pair (M, M') from DifSub, and recording by $false \wedge flag$ that a counterexample was found (line 15).

If EntailsOne does not fail, then a recursive call is performed (line 16) removing the pair (M, M') from DifSub, and recording by $true \wedge flag$ that a counterexample was not found.

If subexpressions M, and M′ have different types (line 17), then it is indicated that the subexpressions have not the same syntactic structure, because NL cannot reason with this kind of expressions, in this case the algorithm finishes returning $false$ (line 18).

When each pair of DifSub has been processed, the result of EntailsAll has to do with whether or not counterexamples have been found (lines 1 and 2).

EntailsOne(M, M')

```
1  switch
2     case polarity(M) = + :
3           return [[M]]_α ⊆ [[M′]]_α
4     case polarity(M) = − :
5           return [[M′]]_α ⊆ [[M]]_α
6     case default :
7           return false
8  end
```

The algorithm EntailsOne codes almost directly Definition 4.1.10, it takes subexpressions M and M′, analysing if their meanings have some containment relationship according to their polarity.

4.4 Examples

At this time, we have a prototype that constructs a model for the pair text-hypothesis of natural language expressions. It is implemented in a strong typed functional programming language called Meta Language (ML), for what has been discussed previously, the prototype asks the user for the veracity of the constructed models in the entailment process.

Example 4.4.1. *An elk ran ⊨ An animal moved*

```
- modChckNatLog([lxe("An",sl(u,sl(u,t,sl(n,t,e)),sl(n,t,e))),
lxe("elk",sl(n,t,e)),lxe("ran",sl(n,t,e))],
[lxe("An",sl(u,sl(u,t,sl(n,t,e)),sl(n,t,e))),lxe("animal",sl(n,t,e)),
```

```
lxe("moved",sl(n,t,e))]);

Is "ran" a kind of, or a manner of "moved"? true
Is "elk" a kind of, or a manner of "animal"? true
The entailment is true.
```

This exemplifies that a very specific statement can be generalized at the extreme that it loses information. Nevertheless, the entailment is true. □

Example 4.4.2. *Dobie brought every ball ⊨ Dobie brought every black ball*

```
- modChckNatLog([lxe("Dobie",sl(u,t,sl(n,t,e))),
lxe("brought",sl(n,sl(n,t,e),e)),
lxe("every",sl(d,bs(u,sl(n,sl(n,t,e),e),sl(n,t,e)),sl(n,t,e))),
lxe("ball",sl(n,t,e))],
[lxe("Dobie",sl(u,t,sl(n,t,e))),lxe("brought",sl(n,sl(n,t,e),e)),
lxe("every",sl(d,bs(u,sl(n,sl(n,t,e),e),sl(n,t,e)),sl(n,t,e))),
lxe("black",sl(u,sl(n,t,e),sl(n,t,e))),lxe("ball",sl(n,t,e))]);

Is "black ball" a kind of, or a manner of "ball"? true
The entailment is true.
```

In this case we have the role of ”every” that is downward monotone on its first argument, therefore it sets the polarity of *ball* to negative. Hence it can be replaced with *black ball* a expression with a lesser denotation. The entailment is true. □

Example 4.4.3. *Dobie didn't bring every ball ⊨ Dobie didn't bring every black ball*

```
- modChckNatLog([lxe("Dobie",sl(u,t,sl(n,t,e))),
lxe("didn't",sl(d,sl(n,t,e),sl(n,t,e))),
lxe("bring",sl(n,sl(n,t,e),e)),
lxe("every",sl(d,bs(u,sl(n,sl(n,t,e),e),sl(n,t,e)),sl(n,t,e))),
lxe("ball",sl(n,t,e))],
[lxe("Dobie",sl(u,t,sl(n,t,e))),lxe("didn't",sl(d,sl(n,t,e),sl(n,t,e))),
lxe("bring",sl(n,sl(n,t,e),e)),
lxe("every",sl(d,bs(u,sl(n,sl(n,t,e),e),sl(n,t,e)),sl(n,t,e))),
lxe("black",sl(u,sl(n,t,e),sl(n,t,e))),lxe("ball",sl(n,t,e))]);

Is "ball" a kind of, or a manner of "black ball"? false
The entailment is false.
```

In this case the verb is negated, therefore it changes the polarity of the following constituents. Hence, the prototype asks if *ball* is a kind of *black ball*, i.e, if the denotation of *ball* is a subset of the denotation of *black ball*. Thence the entailment is false. □

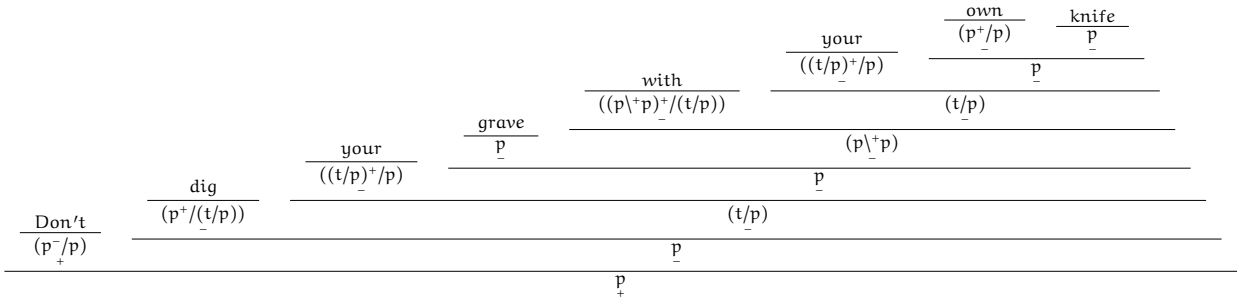


Figure 4.6: Proof tree with polarity marks for the sentence *Don't dig your grave with your own knife*, in this tree $p = (t/e)$.

Example 4.4.4. *Don't dig your grave with your own knife* ⊨ *Don't trench your grave with your own penknife*. For this example, refer to Figure 4.6.

```
- modChckNatLog([lxe("Don't",sl(d,q,q)),lxe("dig",sl(u,q,sl(n,t,q))),
lxe("your",sl(u,sl(n,t,q),q)),lxe("grave",q),
lxe("with",sl(u,bs(u,q,q),sl(n,t,q))),lxe("your",sl(u,sl(n,t,q),q)),
lxe("own",sl(u,q,q)),lxe("knife",q)],
[lxe("Don't",sl(d,q,q)),lxe("trench",sl(u,q,sl(n,t,q)),
lxe("your",sl(u,sl(n,t,q),q)),lxe("grave",q),
lxe("with",sl(u,bs(u,q,q),sl(n,t,q))),lxe("your",sl(u,sl(n,t,q),q)),
lxe("own",sl(u,q,q)),lxe("penknife",q)]);

Is "penknife" a kind of, or a manner of "knife"? true
Is "trench" a manner of "dig"? true
The entailment is true.
```

If WordNet is consulted we find that *trench* is a direct troponym of *dig*. Also, that *penknife* is a hyponym of *knife*. The entailment is true. □

Example 4.4.5. *Don't dig your grave with your own knife* ⊨ *Don't trench your hole with your own penknife*

```
- modChckNatLog([lxe("Don't",sl(d,q,q)),lxe("dig",sl(u,q,sl(n,t,q))),
```

```
lxe("your",sl(u,sl(n,t,q),q)),lxe("grave",q),
lxe("with",sl(u,bs(u,q,q),sl(n,t,q))),lxe("your",sl(u,sl(n,t,q),q)),
lxe("own", sl(u,q,q)), lxe("knife",q)],
[lxe("Don't",sl(d,q,q)),lxe("trench",sl(u,q,sl(n,t,q))),
lxe("your",sl(u,sl(n,t,q),q)),lxe("hole",q),
lxe("with",sl(u,bs(u,q,q),sl(n,t,q))),lxe("your",sl(u,sl(n,t,q),q)),
lxe("own",sl(u,q,q)),lxe("penknife",q)]);

Is "penknife" a kind of, or a manner of "knife"? true
Is "hole" a kind of, or a manner of "grave"? false
Is "trench" a manner of "dig"? true
The entailment is false.
```

penknife is a hyponym of *knife*, *trench* is a direct troponym of *dig*, but *hole* is an hypernym of *grave*, it is not an hyponym. The entailment is false. □

Chapter 5

Pronominal Anaphora Resolution as Parsing

Our proposal to solve pronominal anaphora is inspired by the obvious algorithm commented in section 2.3. *AB* grammars and phrase structure rules, such as used by Kamp and Reyle [KR93], are related to look for possible antecedents in each sentence of a discourse, so some features of phrase structure rules are added to *AB* grammars in order to bind anaphors with antececents.

Conditional sentences and relative clauses are added to *AB* grammars, in such a way that our proposal can resolve Donkey, intra and intersentential anaphora. A method based on model checking is developed to resolve pronominal anaphora.

5.1 Phrase Structure Rules

Pronominal Anaphora is the linguistic phenomenon in which a pronoun points back to a noun phrase. The pronoun is called anaphor and the pointed noun phrase is called antecedent. To solve pronominal anaphora is relevant because it guarantees the cohesion among the sentences of a discourse.

In [KR93], Kamp and Reyle use a phrase structure grammar to carry out the syntactic analysis of English sentences. From the features that they use, we are interested in those related with number (Num = $sing/plur$), gender (Gen = $male/female/-hum$), case ($Case$ = $+nom/-nom$), and a feature to mark if a verb is transitive or intransitive ($Trans$ = $+/-$).

Grammatical categories in phrase structure grammars have not a type, and types in *AB* grammars have no features, therefore we are going to relate both formalisms, after this we will be able to propose a method for anaphora resolution.

Let us begin with the constituents of a sentence, it is formed with a noun phrase

followed by a verb phrase.

$$S_{Num=\alpha} \rightarrow \begin{array}{c} NP \\ \left[\begin{array}{l} Num = \alpha \\ Gen = \beta \\ Case = +nom \end{array}\right] \end{array} \quad VP_{Num=\alpha}$$

As it can be noted, the grammatical categories S and VP only have the feature Num, it is supposed that the Num of S and the Num of VP are equal, this is denoted by writing that both have $Num = \alpha$.

It can also be noted that NP has three features, Num, Gen, and $Case$. Its Num feature has to be α, matching with the type of VP, the gender Gen of NP has no restriction, but the $Case$ feature has to be $+nom$, which means that only nominative pronouns (I, he, she, it, we, you, they) can be in the subject position.

Non nominative pronouns (me, him, her, it, us, you, them) denoted by $Case = -nom$, called objective pronouns, take the place of the object of transitive verbs. It is worth noting that not all the forms of NP have the feature $Case$, hence it is not mandatory that the whole list of features of a grammatical category be present in each occurrence of it.

With these elements it can be said that: a sentence is formed with a noun phrase NP followed by a verb phrase VP, they must be equal in their number, gender does not matter, if the subject is a pronoun, then it must be nominative, finally the sentence inherits the number from the noun phrase and the verb phrase.

What we are going to do is subindexing the corresponding types of *AB* grammars with the features required to get the agreement of number, gender, case and transitivity. From Figure 5.1 we get[1] that the type of noun phrases is $(t/(e\backslash t))$ and that the type of intransitive verbs (the simplest form of a verb phrase) is $(e\backslash t)$, when the most general case for verb phrases is discussed, it will be proved that indeed $(e\backslash t)$ is the type of verb phrases.

In attaching the appropriate subindexes to both types, we get the following proof tree.

$$\frac{(t/(e\backslash t))_{Num=\alpha,Gen=\beta,Case=+nom} \qquad (e\backslash t)_{Num=\alpha}}{t_{Num=\alpha}}$$

Of course, the intuitive meaning of the subindexes is the same as the one for the rules in phrase structure grammars. Note that after a functor for noun phrases is applied to a functor for verb phrases, t is got, saying that the expression analyzed is a sentence in English.

[1] If it is needed to know the type of a grammatical category, Figure 5.1 has to be consulted.

- sentences: t
- proper names: e
- common nouns: (t/e) or r
- intransitive verbs: $(e\backslash t)$ or q
- adjectives: $((t/e)/(t/e))$ or (r/r)
- determiners:
  - at the beginning of a sentence: $((t/(e\backslash t))/(t/e))$ or $((t/q)/r)$
  - in the direct object position of a transitive verb: $((((e\backslash t)/e)\backslash(e\backslash t))/(t/e))$ or $(((q/e)\backslash q)/r)$
- noun phrases: $(t/(e\backslash t))$ or (t/q)
- transitive verbs:
  - with proper names: $((e\backslash t)/e)$ or (q/e)
  - with pronouns: $((e\backslash t)/(t/(e\backslash t)))$ or $(q/(t/q))$
- adverbs: $((e\backslash t)\backslash(e\backslash t))$ or $(q\backslash q)$
- modifiers:
  - for adjectives: $(((t/e)/(t/e))/((t/e)/(t/e)))$ or $((r/r)/(r/r))$
  - for adverbs: $(((e\backslash t)\backslash(e\backslash t))\backslash((e\backslash t)\backslash(e\backslash t)))$ or $((q\backslash q)\backslash(q\backslash q))$
- prepositions:
  - for common noun: $(((t/e)\backslash(t/e))/(t/(e\backslash t)))$ or $((r\backslash r)/(t/q))$
  - for verbs: $(((e\backslash t)\backslash(e\backslash t))/(t/(e\backslash t)))$ or $((q\backslash q)/(t/q))$
- prepositional phrases:
  - for common nouns: $((t/e)\backslash(t/e))$ or $(r\backslash r)$
  - for verbs: $((e\backslash t)\backslash(e\backslash t))$ or $(q\backslash q)$

Figure 5.1: The types of some Grammatical Categories.

Later, a rule stating that a personal noun (type e) is a noun phrase (type $(t/(e\backslash t))$) will be introduced, but in practice the lexicon states that a personal noun has type e, so it is usual to find trees as the next one.

$$\frac{e_{Num=\alpha,Gen=\beta} \qquad (e\backslash t)_{Num=\alpha}}{t_{Num=\alpha}}$$

The following rule states that a noun phrase is composed of a determiner followed by a common noun, the components must agree in number, the noun phrase inherits the number and the gender of the common noun.

$$\begin{array}{ccc} NP & \rightarrow DET_{Num=\alpha} & N \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} & & \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} \end{array}$$

The type of common nouns is (t/e), and for the type of the determiner there exist two cases.

In the first case the determiner is at the beginning of a sentence, in this case its type is $((t/(e\backslash t))/(t/e))$, the following tree is obtained.

$$\frac{((t/(e\backslash t))/(t/e))_{Num=\alpha} \qquad (t/e)_{Num=\alpha,Gen=\beta}}{(t/(e\backslash t))_{Num=\alpha,Gen=\beta}}$$

After the functor for determiners is applied to the functor for common nouns, the type of noun phrases is obtained, and it is ready to be applied to a verb phrase on the right to get a sentence.

The second case is when the determiner is at the beginning of the direct object of a transitive verb, i.e. the determiner is after the verb of the sentence, in which case it has type $((((e\backslash t)/e)\backslash(e\backslash t))/(t/e))$, we have the following tree.

$$\frac{((((e\backslash t)/e)\backslash(e\backslash t))/(t/e))_{Num=\alpha} \qquad (t/e)_{Num=\alpha,Gen=\beta}}{(((e\backslash t)/e)\backslash(e\backslash t))_{Num=\alpha,Gen=\beta}}$$

The root of the tree does not have the type of a noun phrase, but there is no problem, the functor $(((e\backslash t)/e)\backslash(e\backslash t))$ will combine on the left with a transitive verb $((e\backslash t)/e)$ becoming an intransitive verb $(e\backslash t)$.

The next rule states that a personal noun is also a noun phrase.

$$\begin{array}{ccc} NP & \rightarrow & PN \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} & & \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} \end{array}$$

G If an expression occurs in type (b/a), then it may also occur in any type $((b/c)/(a/c))$ (for arbitrary c).

M If an expression occurs in type a, then it may also occur in any type $(b/(b/a))$ (for arbitrary b).

Homomorphic inflation If an expression occurs in type (a/b), then it may also occur in any type $((t/b)/(t/a))$.

Determiners in direct object position If an expression occurs in type $((t/(t/e))/(t/e))$, then it may also occur in any type $(((t/e)/((t/e)/e))/(t/e))$.

Figure 5.2: Rules for change of type in categorial grammars.

Personal nouns have type e, but for the M rule[2] we get $(t/(e\backslash t))$ the type of noun phrases.

Pronouns are also noun phrases, as it was stated, pronouns introduce the feature $Case$.

$$\begin{array}{ccc} NP & \rightarrow & PRO \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \\ Case = \gamma \end{bmatrix} & & \begin{bmatrix} Num = \alpha \\ Gen = \beta \\ Case = \gamma \end{bmatrix} \end{array}$$

If pronouns are noun phrases, then they have type $(t/(e\backslash t))$.

An adjective goes before a common noun, the result is a common noun.

$$\begin{array}{ccc} N & \rightarrow ADJ & N \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} & & \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} \end{array}$$

The type for adjectives is $((t/e)/(t/e))$ and the type for common nouns is (t/e), getting as a result (t/e), a common noun.

$$\frac{((t/e)/(t/e)) \quad (t/e)_{Num=\alpha,Gen=\beta}}{(t/e)_{Num=\alpha,Gen=\beta}}$$

[2] In categorial grammar there exists a serie of rules making possible that certain types can be changed for related types, see Figure 5.2.

A prepositional phrase goes after a common noun giving as result a common noun.

$$\begin{matrix} N \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} \end{matrix} \rightarrow \begin{matrix} N \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} \end{matrix} \quad PP$$

The type of common nouns is (t/e) and the type of prepositional phrases is $((t/e)\backslash(t/e))$.

$$\frac{(t/e)_{Num=\alpha,Gen=\beta} \qquad ((t/e)\backslash(t/e))}{(t/e)_{Num=\alpha,Gen=\beta}}$$

A prepositional phrase is constructed with a preposition followed by a noun phrase.

$$\begin{matrix} PP \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} \end{matrix} \rightarrow P \quad \begin{matrix} NP \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} \end{matrix}$$

A preposition has type $(((t/e)\backslash(t/e))/(t/(e\backslash t)))$ and a noun phrase has type $(t/(e\backslash t))$.

$$\frac{(((t/e)\backslash(t/e))/(t/(e\backslash t))) \qquad (t/(e\backslash t))_{Num=\alpha,Gen=\beta}}{((t/e)\backslash(t/e))_{Num=\alpha,Gen=\beta}}$$

A verb phrase is an intransitive verb.

$$VP_{Num=\alpha} \rightarrow \begin{matrix} V \\ \begin{bmatrix} Num = \alpha \\ Trans = - \end{bmatrix} \end{matrix}$$

Its type in this case is $(e\backslash t)$.

Also, a verb phrase is composed of a transitive verb followed by a noun phrase. In this case note that: the number of the verb and the number of the noun phrase do not have to be equal, the verb has to be transitive, and if the noun phrase is a pronoun, it has to be non nominative.

$$VP_{Num=\alpha} \rightarrow \begin{matrix} V \\ \begin{bmatrix} Num = \alpha \\ Trans = + \end{bmatrix} \end{matrix} \quad \begin{matrix} NP \\ \begin{bmatrix} Num = \beta \\ Gen = \gamma \\ Case = -nom \end{bmatrix} \end{matrix}$$

As it was already stated, we want that the type resulting from applying a transitive verb to a noun phrase to be $(e\backslash t)$. The simplest case is when the transitive verb is applied

to a proper name with type e, knowing that transitive verbs have type $((e\backslash t)/e)$, the following tree is got.

$$\frac{((e\backslash t)/e)_{Num=\alpha,Trans=+} \qquad e_{Num=\beta,Gen=\gamma}}{(e\backslash t)_{Num=\alpha}}$$

When the noun phrase is a pronoun its type is $(t/(e\backslash t))$, the type of transitive verbs should be $((e\backslash t)/(t/(e\backslash t)))$.

$$\frac{((e\backslash t)/(t/(e\backslash t)))_{Num=\alpha,Trans=+} \qquad (t/(e\backslash t))_{Num=\beta,Gen=\gamma,Case=-nom}}{(e\backslash t)_{Num=\alpha}}$$

The problem here is that the type of transitive verbs is $((e\backslash t)/e)$, but by the homomorphic inflation rule (see Figure 5.2) we get $((e\backslash t)/(t/(e\backslash t)))$.

As it was noted, when the direct object of the transitive verb begins with a determiner, the type of the noun phrase is $(((e\backslash t)/e)\backslash(e\backslash t))$ in which case the resulting type is $(e\backslash t)$.

$$\frac{((e\backslash t)/e)_{Num=\alpha,Trans=+} \qquad (((e\backslash t)/e)\backslash(e\backslash t))_{Num=\beta,Gen=\gamma,Case=-nom}}{(e\backslash t)_{Num=\alpha}}$$

Therefore, as it was anticipated, the type of verb phrases is $(e\backslash t)$, hence it does not matter whether the involved verb is transitive or intransitive.

Now, the so called lexical introduction rules are presented, following each rule is shown the corresponding indexed type.

- $\begin{bmatrix} N \\ \begin{bmatrix} Num = sing \\ Gen = male \end{bmatrix} \end{bmatrix}$ → stockbroker, man, . . . ,

  $(t/e)_{Num=sing,Gen=male}$

- $\begin{bmatrix} N \\ \begin{bmatrix} Num = sing \\ Gen = female \end{bmatrix} \end{bmatrix}$ → stockbroker, woman, widow, . . . ,

  $(t/e)_{Num=sing,Gen=female}$

- $\begin{bmatrix} N \\ \begin{bmatrix} Num = sing \\ Gen = -hum \end{bmatrix} \end{bmatrix}$ → book, donkey, horse, Porsche, bicycle, . . . ,

  $(t/e)_{Num=sing,Gen=-hum}$

- $P \rightarrow$ about, behind, except, on, since, toward, ...,
  $(((t/e)\backslash(t/e))/(t/(e\backslash t)))$
- $\text{PRO} \rightarrow$ he,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Gen} = male \\ \text{Case} = +nom \end{bmatrix}$
  $(t/(e\backslash t))_{\text{Num=sing,Gen=male,Case=+nom}}$
- $\text{PRO} \rightarrow$ him,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Gen} = male \\ \text{Case} = -nom \end{bmatrix}$
  $(t/(e\backslash t))_{\text{Num=sing,Gen=male,Case=-nom}}$
- $\text{PRO} \rightarrow$ she,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Gen} = female \\ \text{Case} = +nom \end{bmatrix}$
  $(t/(e\backslash t))_{\text{Num=sing,Gen=female,Case=+nom}}$
- $\text{PRO} \rightarrow$ her,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Gen} = female \\ \text{Case} = -nom \end{bmatrix}$
  $(t/(e\backslash t))_{\text{Num=sing,Gen=female,Case=-nom}}$
- $\text{PRO} \rightarrow$ it,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Gen} = -hum \\ \text{Case} = -nom/+nom \end{bmatrix}$
  $(t/(e\backslash t))_{\text{Num=sing,Gen=-hum,Case=-nom/+nom}}$
- $\text{PRO} \rightarrow$ they,
  $\begin{bmatrix} \text{Num} = plur \\ \text{Gen} = male/female/-hum \\ \text{Case} = +nom \end{bmatrix}$
  $(t/(e\backslash t))_{\text{Num=plur,Gen=male/female/-hum,Case=+nom}}$

- PRO → them,
  $\begin{bmatrix} \text{Num} = plur \\ \text{Gen} = male/female/-hum \\ \text{Case} = -nom \end{bmatrix}$
  $(t/(e\backslash t))_{\text{Num=plur,Gen=male/female/−hum,Case=−nom}}$
- PN → Jones, Smith, Bill, . . . ,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Gen} = male \end{bmatrix}$
  $e_{\text{Num=sing,Gen=male}}$
- PN → Jones, Smith, Mary, Anna Karenina, Madame Bovary, . . . ,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Gen} = female \end{bmatrix}$
  $e_{\text{Num=sing,Gen=female}}$
- PN → Anna Karenina, Buddenbrooks, Middlemarch, Ulyses, . . . ,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Gen} = -hum \end{bmatrix}$
  $e_{\text{Num=sing,Gen=−hum}}$
- V → likes, loves, abhors, rotates, . . . ,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Trans} = + \end{bmatrix}$
  $((e\backslash t)/e)_{\text{Num=sing,Trans=+}}$
- V → stinks, rotates, . . . ,
  $\begin{bmatrix} \text{Num} = sing \\ \text{Trans} = - \end{bmatrix}$
  $(e\backslash t)_{\text{Num=sing,Trans=−}}$

5.2 A Model Checker for Pronominal Anaphora Resolution

In resolving pronominal anaphora, it has to be taken into account that anaphors are pronouns pointing back to proper names, common nouns and noun phrases. But the phenomenon of pointing back can reach sentences that are before the one in which an anaphor appears.

Let $S_1, S_2, \cdots, S_n$, be $n \geq 1$ be sentences in English, with the following configuration S_C:

$$S_C = \begin{cases} S_1 = w_{1,1} : X_{features_{1,1}} w_{2,1} : X_{features_{2,1}} \cdots w_{m_1,1} : X_{features_{m_1,1}}. \\ \vdots \\ S_j = w_{1,j} : X_{features_{1,j}} w_{2,j} : X_{features_{2,j}} \cdots w_{m_j,j} : X_{features_{m_j,j}}. \\ \vdots \\ S_n = w_{1,n} : X_{features_{1,n}} w_{2,n} : X_{features_{2,n}} \cdots w_{m_n,n} : X_{features_{m_n,n}}. \end{cases}$$

As it was shown in the previous section, there exist only lexical insertion rules for proper nouns, common nouns and pronouns. Therefore, it is necesary to look for these grammatical categories in each sentence. Proper and common nouns will be part of the list of *Antecedents*, pronouns will be part of the list of *Anaphors*. This is the purpose of the algorithm FindPCPRON(S_C).

```
Procedure FindPCPRON(S_C)
 1  for j ← 1 to n
 2  do for i ← 1 to m_j
 3      do if w_{i,j} : e_{Num=α,Gen=β}  /*proper names*/
 4            then insert((w_{i,j} : e_{Num=α,Gen=β}, j), Antecedents)
 5         if w_{i,j} : (t/e)_{Num=α,Gen=β}  /*common nouns*/
 6            then insert((w_{i,j} : (t/e)_{Num=α,Gen=β}, j), Antecedents)
 7         if w_{i,j} ∈ {he, him, she, her, it, they, them} /*pronouns*/
 8            then insert((w_{i,j} : (t/(e\t))_{Num=α,Gen=β,Case=γ}, j), Anaphors)
 9         if w_{i,j} ∈ {who, that, which, whose, where, when} /*relative pronouns*/
10            then insert((w_{i,j} : X_{Num=α,Gen=β}, j), Anaphors)
11      [[S_j]]
12  end
```

Let us remember the definition for the meaning of functor application.

Definition 5.2.1. *Meaning of functor application.* Let $X, Y \in L$, and let Z_1, and Z_2 be sequences of types (possibly empty), then

$$[\![Z_1\ (X^s/Y)\ Y\ Z_2]\!] = [\![Z_1\ Y\ (Y\backslash^s X)\ Z_2]\!] = [\![Z_1\ X\ Z_2]\!].$$

□

This definition is fine in general, but it is necessary to distinguish when a noun phrase is formed, when a common noun is after an adjective, and when a common noun is before a prepositional phrase. In other words, it has to be recorded when it is formed an *Antededent* that does not have a lexical insertion rule.

The first case is:

$$\begin{matrix} \mathrm{NP} \\ \begin{bmatrix} \mathrm{Num} = \alpha \\ \mathrm{Gen} = \beta \end{bmatrix} \end{matrix} \rightarrow \mathrm{DET}_{\mathrm{Num}=\alpha} \begin{matrix} \mathrm{N} \\ \begin{bmatrix} \mathrm{Num} = \alpha \\ \mathrm{Gen} = \beta \end{bmatrix} \end{matrix},$$

but this has two subcases.

Determiners have type $((t/(e\backslash t))/(t/e))$ when they are at the beginning of a sentence, so we have:

$$[\![Z_1\ w_{i,j} : ((t/(e\backslash t))/(t/e))_{\mathrm{Num}=\alpha}\ w_{i+1,j} : (t/e)_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : (t/(e\backslash t))_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}\ Z_2]\!] \text{ and}$$
$$\mathit{insert}((w_{i,j}w_{i+1,j} : (t/(e\backslash t))_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}, j), \mathit{Antecedents})$$

The other subcase is when determiners have type $((((e\backslash t)/e)\backslash(e\backslash t))/(t/e))$ because they are in the direct object position of a transitive verb.

$$[\![Z_1\ w_{i,j} : ((((e\backslash t)/e)\backslash(e\backslash t))/(t/e))_{\mathrm{Num}=\alpha}\ w_{i+1,j} : (t/e)_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : (((e\backslash t)/e)\backslash(e\backslash t))_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}\ Z_2]\!] \text{ and}$$
$$\mathit{insert}((w_{i,j}w_{i+1,j} : (((e\backslash t)/e)\backslash(e\backslash t))_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}, j), \mathit{Antecedents})$$

The second case is:

$$\begin{matrix} \mathrm{N} \\ \begin{bmatrix} \mathrm{Num} = \alpha \\ \mathrm{Gen} = \beta \end{bmatrix} \end{matrix} \rightarrow \mathrm{ADJ} \begin{matrix} \mathrm{N} \\ \begin{bmatrix} \mathrm{Num} = \alpha \\ \mathrm{Gen} = \beta \end{bmatrix} \end{matrix},$$

and its corresponding function application is

$$[\![Z_1\ w_{i,j} : ((t/e)/(t/e))\ w_{i+1,j} : (t/e)_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : (t/e)_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}\ Z_2]\!] \text{ and}$$
$$\mathit{insert}((w_{i,j}w_{i+1,j} : (t/e)_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}, j), \mathit{Antecedents})$$

The third case is:

$$\begin{matrix} \mathrm{N} \\ \begin{bmatrix} \mathrm{Num} = \alpha \\ \mathrm{Gen} = \beta \end{bmatrix} \end{matrix} \rightarrow \begin{matrix} \mathrm{N} \\ \begin{bmatrix} \mathrm{Num} = \alpha \\ \mathrm{Gen} = \beta \end{bmatrix} \end{matrix} \mathrm{PP},$$

and it is evaluated in the following way.

$$[\![Z_1\ w_{i,j} : (t/e)_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}\ w_{i+1,j} : ((t/e)\backslash(t/e))\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : (t/e)_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}\ Z_2]\!] \text{ and}$$
$$\mathit{insert}((w_{i,j}w_{i+1,j} : (t/e)_{\mathrm{Num}=\alpha,\mathrm{Gen}=\beta}, j), \mathit{Antecedents})$$

The fourth case is:

$$\begin{matrix} PP \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} \end{matrix} \rightarrow P \quad \begin{matrix} NP \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix} \end{matrix}$$

and it is evaluated in the following way.

$$[\![Z_1\ w_{i,j} : (((t/e)\backslash(t/e))/(t/(e\backslash t)))\ w_{i+1,j} : (t/(e\backslash t))_{Num=\alpha,Gen=\beta}\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : ((t/e)\backslash(t/e))_{Num=\alpha,Gen=\beta}\ Z_2]\!] \text{ and}$$
$$insert((w_{i,j}w_{i+1,j} : ((t/e)\backslash(t/e))_{Num=\alpha,Gen=\beta}, j), Antecedents)$$

For the rest of cases, Definition 5.2.1 is redefined in the following way.

Definition 5.2.2. *Meaning of functor application with carry.* Let $X, Y \in L$, and let Z_1, and Z_2 be sequences of types (possibly empty), then

$$[\![Z_1\ exp_i : (X/Y)\ exp_{i+1} : Y\ Z_2]\!] =$$
$$[\![Z_1\ exp_i : Y\ exp_{i+1} : (Y\backslash X)\ Z_2]\!] =$$
$$[\![Z_1\ exp_i exp_{i+1} : X\ Z_2]\!].$$

□

If these functor applications have been performed, it must be clear that, for example, in processing "a very big apple", in the list of *Antecedents* the expressions "apple", "big apple", "very big apple" and "a very big apple" will be found, but for purposes of pronoun anaphora resolution we are interested in the longest of these expressions, deleting the shorter ones. This work is done by Function longestNP(*list*).

```
Function longestNP(list)
 1  switch
 2    case list = [] :
 3         return []
 4    case list = [(e : X_{Num=α,Gen=β}, i)] :
 5         return [(e : X_{Num=α,Gen=β}, i)]
 6    case list = [(e_1 : X1_{Num=α_1,Gen=β_1}, i_1), (e_2 : X2_{Num=α_2,Gen=β_2}, i_2), ...,
 7         (e_n : Xn_{Num=α_n,Gen=β_n}, i_n)] :
 8         if i_1 = i_2
 9           then if α_1 = α_2 and β_1 = β_2 and
10                    subExpList((e_1 : X1_{Num=α_1,Gen=β_1}, i_1),
11                    [(e_2 : X2_{Num=α_2,Gen=β_2}, i_2), ..., (e_n : Xn_{Num=α_n,Gen=β_n}, i_n)])
12                    then longestNP([(e_2 : X2_{Num=α_2,Gen=β_2}, i_2), ...,
13                         (e_n : Xn_{Num=α_n,Gen=β_n}, i_n)])
14         [(e_1 : X1_{Num=α_1,Gen=β_1}, i_1)] ⊎ longestNP([(e_2 : X2_{Num=α_2,Gen=β_2}, i_2), ...,
15         (e_n : Xn_{Num=α_n,Gen=β_n}, i_n)])
16  end
```

Function subExpList(*exp, expList*) verifies if an expression is a subexpression of a list of expressions, taking into account that they should be in the same sentence.

```
Function subExpList(exp, expList)
 1  switch
 2    case expList = [] :
 3         return false
 4    case exp = (e : X_{Num=α,Gen=β}, i) and
 5         expList = [(e_1 : X1_{Num=α_1,Gen=β_1}, i_1), (e_2 : X2_{Num=α_2,Gen=β_2}, i_2), ...,
 6         (e_n : Xn_{Num=α_n,Gen=β_n}, i_n)] :
 7         if i = i_1
 8           then if α = α_1 and β = β_1 and subExp(e, e_1)
 9                     then return true
10           else subExpList((e : X_{Num=α,Gen=β}, i),
11                [(e_2 : X2_{Num=α_2,Gen=β_2}, i_2), ..., (e_n : Xn_{Num=α_n,Gen=β_n}, i_n)])
12         return false
13  end
```

After this process has been carried out, there exist the list of antecedents and the list of the anaphors, they have the following form, respectively.

$$Antecedents = \begin{cases} [(Ante_1 : X_{1_{Num=\alpha_1,Gen=\beta_1}}, i_1), \\ (Ante_2 : X_{2_{Num=\alpha_2,Gen=\beta_2}}, i_2), \\ \vdots \\ (Ante_n : X_{n_{Num=\alpha_n,Gen=\beta_n}}, i_n)] \end{cases}$$

$$Anaphors = \begin{cases} [(Anap_1 : Y_{1_{Num=\alpha'_1,Gen=\beta'_1,Case=\gamma'_1}}, i'_1), \\ (Anap_2 : Y_{2_{Num=\alpha'_2,Gen=\beta'_2,Case=\gamma'_2}}, i'_2), \\ \vdots \\ (Anap_m : Y_{m_{Num=\alpha'_m,Gen=\beta'_m,Case=\gamma'_m}}, i'_m)] \end{cases}$$

Now, the concept of binding an anaphor to an antecedent can be defined.

Definition 5.2.3. *Binding.* The anaphor $(Anap_i : Y_{i_{Num=\alpha'_i,Gen=\beta'_i,Case=\gamma'_i}}, i'_i)$ binds the antecedent $(Ante_j : X_{j_{Num=\alpha_j,Gen=\beta_j}}, i_j)$ if and only if $i_j \leq i'_i$, $\alpha_j = \alpha'_i$, $\beta_j = \beta'_i$. □

The purpose of Function Bind(*Anaphor, Antecedents, Bindings*) is to bind an *Anaphor* to a list of *Antecedents*, when a binding is possible, it is added to the list of *Bindings*.

```
Function Bind(Anaphor, Antecedents, Bindings)
 1  switch
 2    case Antecedents = [] :
 3         return Bindings
 4    case Anaphor = (Anap : (t/(e\t))_{Num=α,Gen=β,Case=γ}, i)  and
 5         Antecedents = [(Ante_1 : X_{1_{Num=α_1,Gen=β_1}}, i_1),
 6         (Ante_2 : X_{2_{Num=α_2,Gen=β_2}}, i_2), ..., (Ante_n : X_{n_{Num=α_n,Gen=β_n}}, i_n)] :
 7         if i_1 ≤ i
 8           then if α_1 = α  and  β_1 = β
 9                   then Bind(Anaphor, [(Ante_2 : X_{2_{Num=α_2,Gen=β_2}}, i_2), ...,
10                        (Ante_n : X_{n_{Num=α_n,Gen=β_n}}, i_n)],
11                        Bindings ⊎ [((Anap : (t/(e\t))_{Num=α,Gen=β,Case=γ}, i),
12                        (Ante_1 : X_{1_{Num=α_1,Gen=β_1}}, i_1))])
13                   else  Bind(Anaphor, [(Ante_2 : X_{2_{Num=α_2,Gen=β_2}}, i_2), ...,
14                        (Ante_n : X_{n_{Num=α_n,Gen=β_n}}, i_n)], Bindings)
15           else  return Bindings
16  end
```

Function ResAna(*Anaphors, Antecedents, Bindings*) processes the whole list of *Anaphors* trying to bind them to the *Antecedents*, it returns a list of *Bindings*.

Function ResAna(*Anaphors*, *Antecedents*, *Bindings*)

switch
  case *Anaphors* = [] :
    return *Bindings*
  case *Anaphors* = $[(Anap_1 : (t/(e\t))_{Num=\alpha'_1,Gen=\beta'_1,Case=\gamma'_1}, i'_1)$,
    $(Anap_2 : (t/(e\t))_{Num=\alpha'_2,Gen=\beta'_2,Case=\gamma'_2}, i'_2), \ldots,$
    $(Anap_m : (t/(e\t))_{Num=\alpha'_m,Gen=\beta'_m,Case=\gamma'_m}, i'_m)]$ and
    Antecedents = $[(Ante_1 : X_{1_{Num=\alpha_1,Gen=\beta_1}}, i_1)$,
    $(Ante_2 : X_{2_{Num=\alpha_2,Gen=\beta_2}}, i_2), \ldots,$
    $(Ante_n : X_{n_{Num=\alpha_n,Gen=\beta_n}}, i_n)]$:
    ResAna($[(Anap_2 : (t/(e\t))_{Num=\alpha'_2,Gen=\beta'_2,Case=\gamma'_2}, i'_2), \ldots,$
    $(Anap_m : (t/(e\t))_{Num=\alpha'_m,Gen=\beta'_m,Case=\gamma'_m}, i'_m)]$, *Antecedents*,
    Bind($(Anap_1 : (t/(e\t))_{Num=\alpha'_1,Gen=\beta'_1,Case=\gamma'_1}, i'_1)$, *Antecedents*, *Bindings*))
end

Example 5.2.4. Let us consider the following sentences configuration.

$$S_C = \begin{cases} S_1 = \text{Jones owns a book on semantics.} \\ S_2 = \text{He uses it.} \end{cases}$$

Types are omitted to save space, but look at Figure 5.3 as a guide for this example.

After the function FindPCPRON(S_C) is applied, we get:

$$Antecedents = \begin{cases} [(\text{Jones} : e_{Num=sing,Gen=male}, 1), \\ (\text{book} : r_{Num=sing,Gen=-hum}, 1), \\ (\text{semantics} : (t/q)_{Num=sing,Gen=-hum}, 1), \\ (\text{on semantics} : (r\backslash r)_{Num=sing,Gen=-hum}, 1), \\ (\text{book on semantics} : r_{Num=sing,Gen=-hum}, 1), \\ (\text{a book on semantics} : ((q/e)\backslash q)_{Num=sing,Gen=-hum}, 1)] \end{cases}$$

$$Anaphors = \begin{cases} [(\text{He} : (t/q)_{Num=sing,Gen=male,Case=+nom}, 2), \\ (\text{it} : (t/q)_{Num=sing,Gen=-hum,Case=-nom/+nom}, 2)] \end{cases}$$

Then, executing function longestNP(*Antecedents*) only the longest antecedents remain.

$$Antecedents = \begin{cases} [(\text{Jones} : e_{Num=sing,Gen=male}, 1), \\ (\text{a book on semantics} : ((q/e)\backslash q)_{Num=sing,Gen=-hum}, 1)] \end{cases}$$

(a) Proof tree for the expression *book on semantics*.

(b) Proof tree for the sentence *Jones owns a book on semantics*, also see Figure 5.3a.

(c) Proof tree for the sentence *He uses it*.

Figure 5.3: Proof trees for Example 5.2.4.

Finally, function ResAna(*Anaphors, Antecedents, []*) should be executed to get:

$$Bindings = \begin{cases} [((He : (t/q)_{Num=sing,Gen=male,Case=+nom}, 2), \\ (Jones : e_{Num=sing,Gen=male}, 1)), \\ ((it : (t/q)_{Num=sing,Gen=-hum,Case=-nom/+nom}, 2), \\ (a\ book\ on\ semantics : ((q/e)\backslash q)_{Num=sing,Gen=-hum}, 1))] \end{cases}$$

□

Example 5.2.5. Let us consider the following sentences configuration.

$$S_C = \begin{cases} S_1 = Jones\ knows\ a\ woman. \\ S_2 = He\ admires\ her. \\ S_3 = She\ fascinates\ him. \end{cases}$$

Types are omitted to save space, but look at Figure 5.4 as a guide for this example. After the function FindPCPRON(S_C) is applied, we get:

$$Antecedents = \begin{cases} [(Jones : e_{Num=sing,Gen=male}, 1), \\ (woman : r_{Num=sing,Gen=fem}, 1), \\ (a\ woman : ((q/e)\backslash q)_{Num=sing,Gen=fem}, 1)] \end{cases}$$

$$Anaphors = \begin{cases} [(He : (t/q)_{Num=sing,Gen=male,Case=+nom}, 2), \\ (her : (t/q)_{Num=sing,Gen=fem,Case=-nom}, 2), \\ (She : (t/q)_{Num=sing,Gen=fem,Case=+nom}, 3), \\ (him : (t/q)_{Num=sing,Gen=male,Case=-nom}, 3)] \end{cases}$$

Then, executing function longestNP(*Antecedents*) only the longest antecedents remain.

$$Antecedents = \begin{cases} [(Jones : e_{Num=sing,Gen=male}, 1), \\ (a\ woman : ((q/e)\backslash q)_{Num=sing,Gen=fem}, 1)] \end{cases}$$

(a) Proof tree for the sentence *Jones knows a woman.*

(b) Proof tree for the sentence *He admires her.*

(c) Proof tree for the sentence *She fascinates him.*

Figure 5.4: Proof trees for example 5.2.5.

Finally, function ResAna(*Anaphors, Antecedents, []*) should be executed to get:

$$\text{Bindings} = \begin{cases} [((\text{He} : (t/q)_{\text{Num=sing,Gen=male,Case=+nom}}, 2), \\ (\text{Jones} : e_{\text{Num=sing,Gen=male}}, 1)), \\ ((\text{her} : (t/q)_{\text{Num=sing,Gen=fem,Case=-nom}}, 2), \\ (\text{a woman} : ((q/e)\backslash q)_{\text{Num=sing,Gen=fem}}, 1)), \\ ((\text{She} : (t/q)_{\text{Num=sing,Gen=fem,Case=+nom}}, 3), \\ (\text{a woman} : ((q/e)\backslash q)_{\text{Num=sing,Gen=fem}}, 1)), \\ ((\text{him} : (t/q)_{\text{Num=sing,Gen=male,Case=-nom}}, 3), \\ (\text{Jones} : e_{\text{Num=sing,Gen=male}}, 1))] \end{cases}$$

□

5.3 Conditional Sentences and Relative Clauses

To capture conditional sentences in DRT the following phrase structural rule $S \rightarrow$ If S then S is used, we are going to do something similar,

As it can be seen in Figure 5.5, then is followed by S which has type t, and the type of the root is t, therefore the type of then has to be (t/t).

$$\dfrac{\dfrac{\text{then}}{?} \quad \dfrac{S}{t}}{t}$$

Figure 5.5: Looking for the type of *then*.

In Figure 5.6 it can be observed that If needs two arguments, the first one is the sentence after if, and the second is the sentence after then. Hence, the type of If needs to be ((t/t)/t), as it is shown in Figure 5.7.

$$\dfrac{\dfrac{\dfrac{\text{If}}{?} \quad \dfrac{S}{t}}{?} \quad \dfrac{\dfrac{\text{then}}{(t/t)} \quad \dfrac{S}{t}}{t}}{t}$$

Figure 5.6: Looking for the type of *If*.

$$\dfrac{\dfrac{\dfrac{\text{If}}{((t/t)/t)} \quad \dfrac{S}{t}}{(t/t)} \quad \dfrac{\dfrac{\text{then}}{(t/t)} \quad \dfrac{S}{t}}{t}}{t}$$

Figure 5.7: A proof tree for conditional sentences.

Example 5.3.1. Let us consider the following sentence configuration.

$$S_C = \{S_1 = \text{If Jones likes Buddenbrooks then he owns it.}$$

Types are omitted to save space, but look at Figure 5.8 as a guide for this example.

After the function FINDPCPRON(S_C) is applied, we get:

$$\text{Antecedents} = \begin{cases}[(\text{Jones} : e_{\text{Num=sing,Gen=male}}, 1), \\ (\text{Buddenbrooks} : r_{\text{Num=sing,Gen=-hum}}, 1)]\end{cases}$$

$$\text{Anaphors} = \begin{cases}[(\text{He} : (t/q)_{\text{Num=sing,Gen=male,Case=+nom}}, 1), \\ (\text{it} : (t/q)_{\text{Num=sing,Gen=-hum,Case=-nom/+nom}}, 1)]\end{cases}$$

After executing function LONGESTNP(*Antecedents*), it is obtained the same list of Antecedents.

Finally, function RESANA(*Anaphors, Antecedents, []*) should be executed to get:

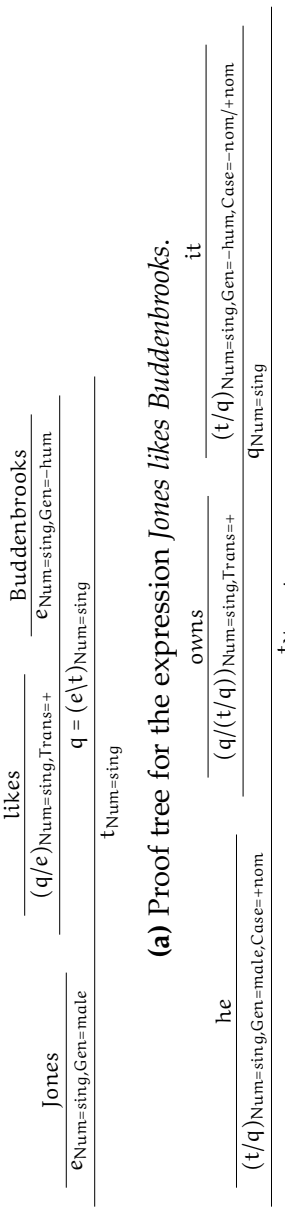
$$\text{Bindings} = \begin{cases}[((\text{He} : (t/q)_{\text{Num=sing,Gen=male,Case=+nom}}, 1), \\ (\text{Jones} : e_{\text{Num=sing,Gen=male}}, 1)), \\ ((\text{it} : (t/q)_{\text{Num=sing,Gen=-hum,Case=-nom/+nom}}, 1), \\ (\text{Buddenbrooks} : r_{\text{Num=sing,Gen=-hum}}, 1))]\end{cases}$$

□

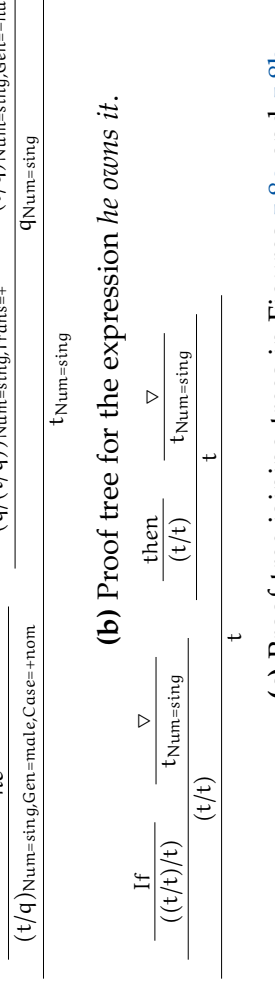
Example 5.3.2. Let us consider the following sentence configuration.

$$S_C = \{S_1 = \text{Every farmer who owns a donkey beats it.}$$

Trying to build a proof tree for it, a failed proof tree is obtained, as it is shown in Figure 5.9. The failed proof tree has three subtrees, the subtree in the middle is a sentence (it has t as root), and if it is taken away, then the left and right subtrees could combine to derive t. □



(a) Proof tree for the expression *Jones likes Buddenbrooks*.



(b) Proof tree for the expression *he owns it*.

(c) Proof tree joining trees in Figures 5.8a and 5.8b.

Figure 5.8: Proof tree for the conditional sentence *If Jones likes Buddenbrooks then he owns it*.

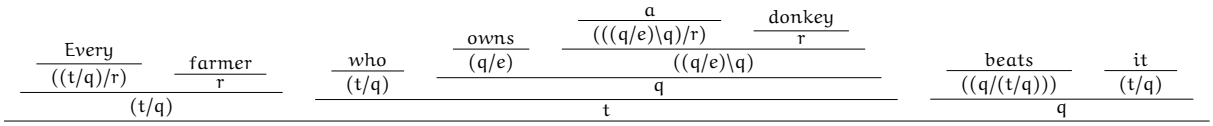


Figure 5.9: A failed proof tree for the sentence *Every farmer who owns a donkey beats it*.

The middle subtree in Figure 5.9 corresponds to a relative clause. This kind of clause begins with the so called relative pronouns *who*, *that*, *which*, *whose*, *where*, *when*, it is frecuently used to identify or define the noun phrase, common noun or proper name preceding it.

Kamp and Reyle [KR93] consider that relative clauses are sentences with a gap. For example, in the expression *owns a donkey* the subject is missing; in the expression *Smith abhors* the direct object is missing. Hence, to fill the gap the empty word ϕ with type (t/q) is introduced. For example, ϕ *owns a donkey*, *Smith abhors* ϕ.

Another aspect to take into account is that relative clauses can refer to the subject or to the direct object of a sentence. If the relative clause refers to the subject of a sentence, then (t/q) is the type of noun phrases with deteminer at the beginning of a sentence, so the type of relative clauses is $((t/q)\backslash(t/q))$.

$$\begin{matrix} \text{NP} \\ \begin{bmatrix} \text{Num} = \alpha \\ \text{Gen} = \beta \end{bmatrix} \end{matrix} \rightarrow \begin{matrix} \text{NP} \\ \begin{bmatrix} \text{Num} = \alpha \\ \text{Gen} = \beta \end{bmatrix} \end{matrix} \quad \begin{matrix} \text{RC} \\ \begin{bmatrix} \text{Num} = \alpha \\ \text{Gen} = \beta \end{bmatrix} \end{matrix}$$

$$\frac{(t/q)_{\text{Num}=\alpha,\text{Gen}=\beta} \quad ((t/q)\backslash(t/q))_{\text{Num}=\alpha,\text{Gen}=\beta}}{(t/q)_{\text{Num}=\alpha,\text{Gen}=\beta}},$$

$((t/q)\backslash(t/q))$ is the type of relative clauses, hence the type of relative pronouns is $(((t/q)\backslash(t/q))/t)$.

$$\begin{matrix} \text{RC} \\ \begin{bmatrix} \text{Num} = \alpha \\ \text{Gen} = \beta \end{bmatrix} \end{matrix} \rightarrow \begin{matrix} \text{RPRO} \\ \begin{bmatrix} \text{Num} = \alpha \\ \text{Gen} = \beta \end{bmatrix} \end{matrix} \quad \begin{matrix} \text{S} \\ \begin{bmatrix} \text{Num} = \alpha \\ \text{Gap} = \text{NP}_{\text{Num}=\alpha} \end{bmatrix} \end{matrix}$$

$$\frac{(((t/q)\backslash(t/q))/t)_{\text{Num}=\alpha,\text{Gen}=\beta} \quad t_{\text{Num}=\alpha,\text{Gap}=\text{NP}_{\text{Num}=\alpha}}}{((t/q)\backslash(t/q))_{\text{Num}=\alpha,\text{Gen}=\beta}},$$

Note that the feature Gap has been added to the grammatical category S. The meaning of $\text{Gap} = \text{NP}_{\text{Num}=\alpha}$ is *the sentence S has a gap in a noun phrase with number* α.

If the relative clause refers to the direct object of a transitive verb, it means that on the left of the relative clause there is a sentence with its own proof tree with root t. Hence, in order to produce a sentence the relative clause has to have type $(t\backslash t)$. Therefore we have the following phrase structure rule with its respective proof tree.

$$S_{Num=\alpha} \rightarrow S_{Num=\alpha} \quad \underset{\begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix}}{RC}$$

$$\frac{t_{Num=\alpha} \quad (t\backslash t)_{Num=\alpha,Gen=\beta}}{t_{Num=\alpha}},$$

The relative clause in this case has type $(t\backslash t)$, S has type t, in such a way that the relative pronoun has type $((t\backslash t)/t)$, as it is shown below.

$$\underset{\begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix}}{RC} \rightarrow \underset{\begin{bmatrix} Num = \alpha \\ Gen = \beta \end{bmatrix}}{RPRO} \quad \underset{\begin{bmatrix} Num = \alpha \\ Gap = NP_{Num=\alpha} \end{bmatrix}}{S}$$

$$\frac{((t\backslash t)/t)_{Num=\alpha,Gen=\beta} \quad t_{Num=\alpha,Gap=NP_{Num=\alpha}}}{(t\backslash t)_{Num=\alpha,Gen=\beta}},$$

The following lexical introduction rule for the empty noun phrase is required.

$$\underset{\begin{bmatrix} Num = \alpha \\ Gen = \beta \\ Case = \gamma \\ Gap = NP_{Num=\alpha} \end{bmatrix}}{NP} \rightarrow \phi$$

As the empty string ϕ is a NP, it has type (t/q). If after a relative pronoun there is a subject, then the gap is at the position of the direct object (after the verb) of the relative clause. If after a relative pronoun there is a verb, then the gap is at the position of the subject (before the verb) of the relative clause.

If the relative pronoun "who" refers to the subject of the main sentence then it has type $(((t/q)\backslash(t/q))/t)_{Num=sing/plur,Gen=male/fem}$; or when it refers to the direct object of the main sentence, it has type $((t\backslash t)/t)_{Num=sing/plur,Gen=male/fem}$.

$$\underset{\begin{bmatrix} Num = sing/plur \\ Gen = male/fem \end{bmatrix}}{RPRO} \rightarrow \text{who}$$

If the relative pronoun "which" refers to the subject of the main sentence then it has type $(((t/q)\backslash(t/q))/t)_{Num=sing/plur,Gen=-hum}$; or when it refers to the direct object of the main sentence, it has type $((t\backslash t)/t)_{Num=sing/plur,Gen=-hum}$.

$$\begin{array}{c} RPRO \\ \begin{bmatrix} Num = sing/plur \\ Gen = -hum \end{bmatrix} \end{array} \quad \rightarrow \text{which}$$

Two phrase structure rules are introduced for the grammatical category S. In the first one, the gap is in the noun phrase of the sentence, the number of the gap must match the number of the noun phrase and the number of the verb phrase. The verb phrase cannot have gap.

$$\begin{array}{c} S \\ \begin{bmatrix} Num = \alpha \\ Gap = NP_{Num=\alpha} \end{bmatrix} \end{array} \rightarrow \begin{array}{c} NP \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \\ Case = +nom \\ Gap = NP_{Num=\alpha} \end{bmatrix} \end{array} \quad \begin{array}{c} VP \\ \begin{bmatrix} Num = \alpha \\ Gap = - \end{bmatrix} \end{array}$$

$$\frac{(t/(e\backslash t))_{Num=\alpha,Gen=\beta,Case=+nom,Gap=NP_{Num=\alpha}} \quad (e\backslash t)_{Num=\alpha,Gap=-}}{t_{Num=\alpha,Gap=NP_{Num=\alpha}}}$$

In the second one, the noun phrase does not have gap, the verb phrase has it, the number of the gap does not have to match the number of the noun phrase or the number of the verb phrase.

$$\begin{array}{c} S \\ \begin{bmatrix} Num = \alpha \\ Gap = NP_{Num=\gamma} \end{bmatrix} \end{array} \rightarrow \begin{array}{c} NP \\ \begin{bmatrix} Num = \alpha \\ Gen = \beta \\ Case = +nom \\ Gap = - \end{bmatrix} \end{array} \quad \begin{array}{c} VP \\ \begin{bmatrix} Num = \alpha \\ Gap = NP_{Num=\gamma} \end{bmatrix} \end{array}$$

$$\frac{(t/(e\backslash t))_{Num=\alpha,Gen=\beta,Case=+nom,Gap=-} \quad (e\backslash t)_{Num=\alpha,Gap=NP_{Num=\gamma}}}{t_{Num=\alpha,Gap=NP_{Num=\gamma}}}$$

Now, we have everything we need to define how the meaning of an expression in natural language is computed, with their corresponding dynamic elements to solve pronominal anaphora. This definition corresponds to line 11 of algorithm FINDPCPRON(S_C).

Definition 5.3.3. *Meaning of functor application for composed sentences.* Let $X, Y \in L$, and let Z_1, and Z_2 be sequences of types (possibly empty), then

1. *NP* → *DET N*, at the beginning of a sentence:

$$[\![Z_1\ w_{i,j} : ((t/(e\backslash t))/(t/e))_{Num=\alpha}\ w_{i+1,j} : (t/e)_{Num=\alpha,Gen=\beta}\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : (t/(e\backslash t))_{Num=\alpha,Gen=\beta}\ Z_2]\!] \text{ and}$$
$$\mathit{insert}((w_{i,j}w_{i+1,j} : (t/(e\backslash t))_{Num=\alpha,Gen=\beta}, j), \mathit{Antecedents})$$

2. *NP* → *DET N*, at the beginning of a direct object:

$$[\![Z_1\ w_{i,j} : ((((e\backslash t)/e)\backslash(e\backslash t))/(t/e))_{Num=\alpha}\ w_{i+1,j} : (t/e)_{Num=\alpha,Gen=\beta}\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : (((e\backslash t)/e)\backslash(e\backslash t))_{Num=\alpha,Gen=\beta}\ Z_2]\!] \text{ and}$$
$$\mathit{insert}((w_{i,j}w_{i+1,j} : (((e\backslash t)/e)\backslash(e\backslash t))_{Num=\alpha,Gen=\beta}, j), \mathit{Antecedents})$$

3. *N* → *ADJ N*

$$[\![Z_1\ w_{i,j} : ((t/e)/(t/e))\ w_{i+1,j} : (t/e)_{Num=\alpha,Gen=\beta}\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : (t/e)_{Num=\alpha,Gen=\beta}\ Z_2]\!] \text{ and}$$
$$\mathit{insert}((w_{i,j}w_{i+1,j} : (t/e)_{Num=\alpha,Gen=\beta}, j), \mathit{Antecedents})$$

4. *N* → *N PP*

$$[\![Z_1\ w_{i,j} : (t/e)_{Num=\alpha,Gen=\beta}\ w_{i+1,j} : ((t/e)\backslash(t/e))\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : (t/e)_{Num=\alpha,Gen=\beta}\ Z_2]\!] \text{ and}$$
$$\mathit{insert}((w_{i,j}w_{i+1,j} : (t/e)_{Num=\alpha,Gen=\beta}, j), \mathit{Antecedents})$$

5. *PP* → *P NP*

$$[\![Z_1\ w_{i,j} : (((t/e)\backslash(t/e))/(t/(e\backslash t)))\ w_{i+1,j} : (t/(e\backslash t))_{Num=\alpha,Gen=\beta}\ Z_2]\!] =$$
$$[\![Z_1\ w_{i,j}w_{i+1,j} : ((t/e)\backslash(t/e))_{Num=\alpha,Gen=\beta}\ Z_2]\!] \text{ and}$$
$$\mathit{insert}((w_{i,j}w_{i+1,j} : ((t/e)\backslash(t/e))_{Num=\alpha,Gen=\beta}, j), \mathit{Antecedents})$$

6. Functor application in any other case.

$$[\![Z_1\ exp_i : (X/Y)\ exp_{i+1} : Y\ Z_2]\!] =$$
$$[\![Z_1\ exp_i : Y\ exp_{i+1} : (Y\backslash X)\ Z_2]\!] =$$
$$[\![Z_1\ exp_i\ exp_{i+1} : X\ Z_2]\!].$$

7. The case for compound sentences, the argument of the functor is on the right, and the functor cannot be applied.

$$[\![Z_1\ exp_i : (X/Y)\ exp_{i+1} : Z\ Z_2]\!] =$$
$$[\![[\![Z_1\ exp_i : (X/Y)]\!]\ exp_{i+1} : Z\ Z_2]\!]$$

8. The case for compound sentences, the argument of the functor is on the left, and the functor cannot be applied.

$$[\![Z_1\ exp_i : Z\ exp_{i+1} : (Y\backslash X)\ Z_2]\!] =$$
$$[\![[\![Z_1\ exp_i : Z]\!]\ exp_{i+1} : (Y\backslash X)\ Z_2]\!]$$

9. The sentence has been reduced to a single expression.

$$[\![exp : X]\!] = exp : X$$

□

Cases 1 to 5 in Definition 5.3.3 are the dynamic behavior given to *AB* grammars. Case 6 is the usual functor application, or the statical functor application rule. The main idea behind cases 7 and 8 is that in parsing a sentence the process is conducted from the right to the left of the sentence, in such a way that when it is not possible to apply a functor, the words on the left of the current parsed subexpression have to be parsed next, hoping that after they are parsed both will combine.

To understand this, it is convenient to do the following analysis, the point here is to understand how a sentence in *AB* grammars is parsed, in other words, how a proof tree is constructed in order to get a sentence.

- A noun phrase followed by a verb phrase, where the verb phrase is an intransitive verb.

  1. The noun phrase is a proper name, there are only two words.

$$\frac{\dfrac{\mathit{anyProperName}}{e} \qquad \dfrac{\mathit{anyIntransitiveVerb}}{q = (e\backslash t)}}{t}$$

  2. The noun phrase is a pronoun, also there are only two words.

$$\frac{\dfrac{\mathit{anyPronoun}}{(t/q)} \qquad \dfrac{\mathit{anyIntransitiveVerb}}{q}}{t}$$

3. The noun phrase begins with a determiner, at least there are three words, it depends of the complexity of the noun.

$$\dfrac{\dfrac{\dfrac{\mathit{anyDet}}{((t/q)/r)} \quad \dfrac{\mathit{anyNoun}}{r}}{(t/q)} \quad \dfrac{\mathit{anyIntransitiveVerb}}{q}}{t}$$

anyNoun cannot combine with anyIntransitiveVerb, so anyDet has to combine first with anyNoun, case 7 in Definition 5.3.3.

- A noun phrase followed by a verb phrase, where the verb phrase is a transitive verb. As it was noted, a verb phrase is always reduced to q, so the interesting thing here is to analyze the structure of the verb phrase, i.e., a transitive verb followed by a noun phrase.

1. The noun phrase is a proper name.

$$\dfrac{\dfrac{\mathit{anyTransitiveVerb}}{(q/e)} \quad \dfrac{\mathit{anyProperName}}{e}}{q}$$

2. The noun phrase is a pronoun.

$$\dfrac{\dfrac{\mathit{anyTransitiveVerb}}{(q/(t/q))} \quad \dfrac{\mathit{anyPronoun}}{(t/q)}}{q}$$

3. The noun phrase begins with a determiner.

$$\dfrac{\dfrac{\mathit{anyTransitiveVerb}}{(q/e)} \quad \dfrac{\dfrac{\mathit{anyDet}}{(((q/e)\backslash q)/r)} \quad \dfrac{\mathit{anyNoun}}{r}}{((q/e)\backslash q)}}{q}$$

From the previous analysis it can be concluded that in order to parse a sentence in *AB* grammars, the process is done from the right to the left of the sentence. In other words, to combine a noun phrase with a verb phrase to get a sentence, the verb phrase has to be parsed first.

Cases 7 or 8 in Definition 5.3.3 are used in parsing sentences beginning with a determiner, sentences containing a relative clause or conditional sentences.

Example 5.3.4. This example shows the behavior of Definition 5.3.3, types and their features are avoided to save space, if you wish, look at Figure 5.10 as a guide for functor

applications. When two words or expressions are underlined it means that a functor application has been applied. Each number over the equal sign corresponds to the case in Definition 5.3.3 used to compute $[\![exp]\!]$.

$$[\![\text{Every farmer who } \phi \text{ owns a donkey beats it}]\!] \overset{6}{=}$$

$$[\![\text{Every farmer who } \phi \text{ owns a donkey } \underline{\text{beats it}}]\!] \overset{6}{=}$$

$$[\![[\![\text{Every farmer who } \phi \text{ owns a donkey}]\!] \underline{\text{beats it}}]\!] \overset{7}{=}$$

$$[\![[\![\text{Every farmer who } \phi \text{ owns } \underline{\text{a donkey}}]\!] \underline{\text{beats it}}]\!] \text{ and}$$

$$\text{insert}(\text{a donkey}, \text{Antecedents}) \overset{2}{=}$$

$$[\![[\![\text{Every farmer who } \phi \underline{\text{ owns a donkey}}]\!] \underline{\text{beats it}}]\!] \text{ and}$$

$$\text{insert}(\text{a donkey}, \text{Antecedents}) \overset{6}{=}$$

$$[\![[\![\text{Every farmer who } \underline{\phi \text{ owns a donkey}}]\!] \underline{\text{beats it}}]\!] \text{ and}$$

$$\text{insert}(\text{a donkey}, \text{Antecedents}) \overset{6}{=}$$

$$[\![[\![\text{Every farmer } \underline{\text{who } \phi \text{ owns a donkey}}]\!] \underline{\text{beats it}}]\!] \text{ and}$$

$$\text{insert}(\text{a donkey}, \text{Antecedents}) \overset{7}{=}$$

$$[\![[\![[\![\text{Every farmer}]\!] \underline{\text{who } \phi \text{ owns a donkey}}]\!] \underline{\text{beats it}}]\!] \text{ and}$$

$$\text{insert}(\text{a donkey}, \text{Antecedents}) \overset{1}{=}$$

$$[\![[\![[\![\underline{\text{Every farmer}}]\!] \underline{\text{who } \phi \text{ owns a donkey}}]\!] \underline{\text{beats it}}]\!] \text{ and}$$

$$\text{insert}(\text{Every farmer}, \text{Antecedents}) \text{ and}$$

$$\text{insert}(\text{a donkey}, \text{Antecedents}) \overset{9}{=}$$

$$[\![[\![\underline{\text{Every farmer}} \underline{\text{ who } \phi \text{ owns a donkey}}]\!] \underline{\text{beats it}}]\!] \text{ and}$$

$$\text{insert}(\text{Every farmer}, \text{Antecedents}) \text{ and}$$

$$\text{insert}(\text{a donkey}, \text{Antecedents}) \overset{6}{=}$$

$$[\![[\![\underline{\text{Every farmer who } \phi \text{ owns a donkey}}]\!] \underline{\text{beats it}}]\!] \text{ and}$$

$$\text{insert}(\text{Every farmer}, \text{Antecedents}) \text{ and}$$

$$\text{insert}(\text{a donkey}, \text{Antecedents}) \overset{9}{=}$$

$$[\![\underline{\text{Every farmer who } \phi \text{ owns a donkey}} \underline{\text{ beats it}}]\!] \text{ and}$$

$$\text{insert}(\text{Every farmer}, \text{Antecedents}) \text{ and}$$

$$\mathit{insert}(\text{a donkey}, \mathit{Antecedents}) \overset{6}{=}$$

$$[\![\underline{\text{Every farmer who } \phi \text{ owns a donkey beats it}}]\!] \text{ and}$$

$$\mathit{insert}(\text{Every farmer}, \mathit{Antecedents}) \text{ and}$$

$$\mathit{insert}(\text{a donkey}, \mathit{Antecedents}) \overset{9}{=}$$

$$\underline{\text{Every farmer who } \phi \text{ owns a donkey beats it}} \text{ and}$$

$$\mathit{insert}(\text{Every farmer}, \mathit{Antecedents}) \text{ and}$$

$$\mathit{insert}(\text{a donkey}, \mathit{Antecedents})$$

The following three examples show how pronominal anaphora is solved with relative clauses.

Example 5.3.5. Let us consider the following sentence configuration.

$$S_C = \{S_1 = \text{Every farmer who owns a donkey beats it.}$$

Types are omitted to save space, but look at Figure 5.10 as a guide for this example.

After the function FindPCPRON(S_C) is applied, we get:

$$\mathit{Antecedents} = \begin{cases} [(\text{farmer} : r_{\text{Num=sing,Gen=male}}, 1), \\ (\text{donkey} : r_{\text{Num=sing,Gen=-hum}}, 1), \\ (\text{a donkey} : ((q/e)\backslash q)_{\text{Num=sing,Gen=-hum}}, 1), \\ (\text{Every farmer} : (t/q)_{\text{Num=sing,Gen=male}}, 1)] \end{cases}$$

$$\mathit{Anaphors} = \begin{cases} [(\text{it} : (t/q)_{\text{Num=sing,Gen=-hum,Case=-nom/+nom}}, 1) \\ (\text{who} : (((t/q)\backslash(t/q))/t)_{\text{Num=sing/plur,Gen=male/fem}}, 1)] \end{cases}$$

Then, executing function longestNP(*Antecedents*) only the longest antecedents remain.

$$\mathit{Antecedents} = \begin{cases} [(\text{a donkey} : ((q/e)\backslash q)_{\text{Num=sing,Gen=-hum}}, 1), \\ (\text{Every farmer} : (t/q)_{\text{Num=sing,Gen=male}}, 1)] \end{cases}$$

Finally, function ResAna(*Anaphors*, *Antecedents*, *[]*) should be executed to get:

$$\mathit{Bindings} = \begin{cases} [((\text{it} : (t/q)_{\text{Num=sing,Gen=-hum,Case=-nom/+nom}}, 1), \\ (\text{a donkey} : ((q/e)\backslash q)_{\text{Num=sing,Gen=-hum}}, 1)), \\ ((\text{who} : (((t/q)\backslash(t/q))/t)_{\text{Num=sing/plur,Gen=male/fem}}, 1), \\ (\text{Every farmer} : (t/q)_{\text{Num=sing,Gen=male}}, 1))] \end{cases}$$

□

Figure 5.10: Proof tree for the sentence *Every farmer who owns a donkey beats it*.

Example 5.3.6. Let us consider the following sentence configuration.

$$S_C = \{S_1 = \text{Mary loves a book which Smith abhors.}$$

Types are omitted to save space, but look at Figure 5.11 as a guide for this example.

After the function FindPCPRON(S_C) is applied, we get:

$$\text{Antecedents} = \begin{cases} [(\text{Mary} : e_{\text{Num=sing,Gen=female}}, 1), \\ (\text{book} : r_{\text{Num=sing,Gen=-hum}}, 1), \\ (\text{Smith} : e_{\text{Num=sing,Gen=male}}, 1), \\ (\text{a book} : ((q/e)\backslash q)_{\text{Num=sing,Gen=-hum}}, 1)] \end{cases}$$

$$\text{Anaphors} = \{[(\text{which} : ((t\backslash t)/t)_{\text{Num=sing/plur,Gen=-hum}}, 1)]$$

Then, executing function longestNP(*Antecedents*) only the longest antecedents remain.

$$\text{Antecedents} = \begin{cases} [(\text{Mary} : e_{\text{Num=sing,Gen=female}}, 1), \\ (\text{Smith} : e_{\text{Num=sing,Gen=male}}, 1), \\ (\text{a book} : ((q/e)\backslash q)_{\text{Num=sing,Gen=-hum}}, 1)] \end{cases}$$

Finally, function ResAna(*Anaphors, Antecedents, []*) should be executed to get:

$$\text{Bindings} = \begin{cases} [((\text{which} : ((t\backslash t)/t)_{\text{Num=sing/plur,Gen=-hum}}, 1), \\ (\text{a book} : ((q/e)\backslash q)_{\text{Num=sing,Gen=-hum}}, 1))] \end{cases}$$

□

Example 5.3.7. Let us consider the following sentence configuration.

$$S_C = \{S_1 = \text{Jones likes a stockbroker who loves Mary.}$$

Types are omitted to save space, but look at Figure 5.12 as a guide for this example.

After the function FindPCPRON(S_C) is applied, we get:

$$\text{Antecedents} = \begin{cases} [(\text{Jones} : e_{\text{Num=sing,Gen=male}}, 1), \\ (\text{stockbroker} : r_{\text{Num=sing,Gen=male/fem}}, 1), \\ (\text{Mary} : e_{\text{Num=sing,Gen=female}}, 1), \\ (\text{a stockbroker} : ((q/e)\backslash q)_{\text{Num=sing,Gen=male/fem}}, 1)] \end{cases}$$

$$\text{Anaphors} = \{[(\text{who} : ((t\backslash t)/t)_{\text{Num=sing/plur,Gen=male/fem}}, 1)]$$

(a) Proof tree for the expression *which Smith abhors*. The features of ϕ (Num = sing/plur, Gen = male/fem/ − hum, Case = −nom, Gap = $NP_{Num=sing/plur}$) are omitted to save space.

(b) Proof tree for the expression *Mary loves a book*.

(c) Proof tree joining trees in Figures 5.11b and 5.11a.

Figure 5.11: Proof tree for the sentence *Mary loves a book which Smith abhors*.

(a) Proof tree for the expression *who loves Mary*. The features of ϕ (Num = sing, Gen = male/fem, Case = +nom, Gap = $NP_{Num=sing}$) are omitted to save space.

(b) Proof tree for the expression *Jones likes a stockbroker*.

(c) Proof tree joining trees in Figures 5.12b and 5.12a.

Figure 5.12: Proof trees for the sentence *Jones likes a stockbroker who loves Mary*.

Then, executing function LONGESTNP(*Antecedents*) only the longest antecedents remain.

$$Antecedents = \begin{cases} [(Jones : e_{Num=sing,Gen=male}, 1), \\ (Mary : e_{Num=sing,Gen=female}, 1), \\ (a\ stockbroker : ((q/e)\backslash q)_{Num=sing,Gen=male/fem}, 1)] \end{cases}$$

Finally, function RESANA(*Anaphors, Antecedents, []*) should be executed to get:

$$Bindings = \begin{cases} [((who : ((t\backslash t)/t)_{Num=sing/plur,Gen=male/fem}, 1), \\ (a\ stockbroker : ((q/e)\backslash q)_{Num=sing,Gen=male/fem}, 1))] \end{cases}$$

□

Chapter 6

Closure

We close this dissertation stating conclusions about the achievements reached, defining an agenda for future work, and summarizing the tangible contributions.

6.1 Conclusions

Shortly, this work has been about the application of formal semantics to the recognition of textual entailment via NL. A model theoretic semantics and a model checker for NL were developed, with these it can be said why a text does not entail a hypothesis, so the explanatory power of NL has been recovered. The open questions about the polarity algorithms have been closed proving their equivalence. Also, the problem of pronominal anaphora resolution was solved by dynamizing the model-theoretic semantics of an extension of *AB* grammars, with this, a model checker for solving pronominal anaphora was defined. Next, detailed conclusions can be found.

The equivalence of the algorithms of van Benthem, Sánchez, Dowty and van Eijck permits to state that the algorithms of van Benthem and van Eijck are sound, also that the semantics of Sánchez and Moss are equivalent. Contrary to Dowty's assumption, his internal algorithm has no advantage with respect to the external algorithms.

As far as we know, the equivalence between the algorithms for marking polarity here analyzed had not been proved, therefore we have proved that the concept of polarity computed by these algorithms is the same.

We are not linguists, but as computer scientists we would prefer the van Benthem's algorithm because it is the simplest and it just needs two steps to compute polarity. The algorithm of Dowty would be the second option, apparently it needs one single step but when the functor is the result of a composition, the algorithm has to make a guess, and if it fails, backtracking is required (this has been shown in the explanation of the construction of the proof tree of Figure 3.5.). The algorithms of Sánchez and van Eijck require three steps.

From the model theoretic semantics here developed, a model checker for Natural Logic has been proposed, and the explanatory power of Natural Logic has been recovered. Using Natural Logic, the recognizable textual entailments are restricted to those sentences that have the same syntactic structure.

The model theoretic semantics is based on the denotations of the subexpressions of a natural language sentence, these denotations just are sets. Hence, a high order semantics is not needed.

To implement a model checker for NL, lexicons having pairs $word : type$ are needed, but, as far as we know, there are not such lexicons. Another possibility is to have a Part Of Speech (POS) tagger that associates each word with its proper type.

The *C & C* tools [CC04] have a POS tagger, but it uses the inference rules of Combinatory Categorial Grammars, and there is not an algorithm to compute polarity for this kind of grammars, actually the known algorithms to compute polarity only work with Categorial Grammars which have function application as the unique inference rule.

There are not domains partially ordered as it is supposed in section 2.2, therefore to compute the meaning of a natural language expression is not possible, but it is possible to take advantage of tools such as WordNet [Uni10], BabelNet [NP12], etc., to find synonyms, hyponyms, hyperonyms, and troponyms.

By relating *AB* grammars with phrase structure grammars it is possible to construct the noun phrase antecedents in parsing time, even though the binding phase between anaphors and antecedents is performed after the parsing time.

Nevertheless, *AB* grammars are a kind of Type Logical Grammar [Jäg05], it should be noticed that our method of anaphora resolution does not produce any logical form, because the syntactic part to look for antecedents is enough, from there the name Anaphora Resolution as Parsing.

It should be noticed that our pronominal anaphora solver is in the line of knowledge poor methods, because it is only necessary the lexical knowledge about the type and features of each word in a sentence, of course also the syntactic knowledge underlying Definition 5.3.3 is needed. In spite of being a poor knowledge method, the method can solve Donkey, inter and intra sentential pronominal anaphora. Unfortunately, there are not lexicons having pairs $word : type_{features}$.

6.2 Future Work

1. In order to widen the scope of NL to Recognize Textual Entailment, it is desirable to be able to compare subexpressions of similar types; for example, the type of nouns

is similar to the type of noun phrases.

2. Another point of interest is to compare the proposals to dynamise higher order logics, looking for an unifying type logical grammar.

3. Other points on the agenda for future work are:

   (a) To construct lexicons where the words are associated with their types and features.

   (b) To define an algorithm to compute polarity for Combinatory Categorial Grammars.

   (c) To build interfaces to take advantage of resources such as WordNet, and BabelNet.

6.3 Contributions

The main contributions of the research reported here are the following:

1. The equivalence among the polarity algorithms presented in Chapter 3 has been proved, which makes clear that internal and external polarity marking compute the same notion of polarity. As others consequences, the soundness of the algorithms of van Benthem and van Eijck are established. The work developed there has been electronically published in *Studia Logica* with the title *Equivalences among polarity algorithms*.

2. A model theoretic semantics for NL has been defined, and it is sound and complete to both, the proof theory of an extension of *AB* grammars and the proof theory of NL. This permits us to define a method of model checking for NL and characterize the kind of textual entailments that can be recognized with NL. All this work is reported in the article *Textual Entailment via Model Checking for NL*. It has been sent to *Studia Logica* for revision.

3. An automatic theorem prover for NL has been developed. With this, it was possible to compute the polarity of the subexpressions of a sentence. This was informed in the article *Automatic Theorem Proving for NL: a Case Study on Textual Entailment*. It has been accepted for publication in *Computación y Sistemas*.

4. A model checker to solve pronominal anaphora has been built. It is a poor knowledge method. It covers Donkey, intrasentential and intersentencial anaphora, and it does not need a representation in logical form. This contribution will be sent for publication to *Computational Linguistics* with the title *Pronominal Anaphora Resolution as Parsing*.

Bibliography

[Ajd78] Kazimierz Ajdukiewicz. Syntactic connexion (1936). In Jerzy Giedymin, editor, *The Scientific World-Perspective and Other Essays, 1931–1963*, pages 118–139. Springer Netherlands, Dordrecht, 1978.

[Akh06] Elena Akhmatova. Textual entailment resolution via atomic propositions. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Textual Entailment: First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, pages 385–403, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Bac88] Emmon Bach. Categorial grammars as theories of language. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*, pages 17–34. Springer Netherlands, Dordrecht, 1988.

[BM06a] Johan Bos and Katja Markert. Recognising textual entailment with robust logical inference. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment: First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, pages 404–426, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BM06b] Johan Bos and Katja Markert. When logical inference helps determining textual entailment (and when it doesn't). In *Proceedings of the Second Challenge Workshop, Recognizing Textual Entailment*, pages 98–103. Pascal, 2006.

[CC04] Stephen Clark and James R. Curran. Parsing the wsj using ccg and log-linear models. In *Proceedings of the 42Nd Annual Meeting on Association for*

Computational Linguistics, ACL '04, pages 103–110, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[dG06] Philippe de Groote. Towards a montagovian account of dynamics. *Semantics and Linguistic Theory*, 16(0):1–16, 2006.

[DGM06] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment: First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, pages 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Dow94] David Dowty. The role of negative polarity and concord marking in natural language reasoning. In Harvey and Santelmann, editors, *SALT IV*, pages 114–145, Ithaca, NY, 1994. Cornel University.

[DRSZ13] Ido Dagan, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. *Recognizing Textual Entailment: Models and Applications*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2013.

[Eij99] J. van Eijck. Axiomatising dynamic logics for anaphora. *Journal of Language and Computation*, 1:103–126, 1999.

[Gia11] Anastasia Giannakidou. Positive polarity items and negative polarity items: Variation, licensing, and compositionality. In Claudia Maienborn, Klaus von Heusinger, and Paul Portner, editors, *Semantics: An International Handbook of Natural Language Meaning*, pages 1660–1712. De Gruyter Mouton, 2011.

[GS89] Jeroen Groenendijk and Martin Stokhof. Dynamic montague grammar. In *Papers from the Second Symposium on Logic and Language*, pages 3–48. Akademiai Kiadoo, 1989.

[GS91] Jeroen Groenendijk and Martin Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14(1):39–100, 1991.

[HCFM06] Daniel Hodges, Christine Clark, Abraham Fowler, and Dan Moldovan. Applying cogex to recognize textual entailment. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine*

Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Textual Entailment: First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers, pages 427–448, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[HTK00] David Harel, Jerzy Tiuryn, and Dexter Kozen. *Dynamic Logic*. MIT Press, Cambridge, MA, USA, 2000.

[IIM14] Thomas F. Icard III and Lawrence S. Moss. Recent progress on monotonicity. *Linguistic Issues in Language Technology, Perspectives on Semantic Representations for Textual Inference*, 9:167–194, 2014.

[Isr11] M. Israel. *The Grammar of Polarity: Pragmatics, Sensitivity, and the Logic of Scales*. Cambridge Studies in Linguistics. Cambridge University Press, 2011.

[Jäg05] Gerhard Jäger. *Anaphora and Type Logical Grammar*. Trends in Logic 24. Springer, Dordrecht, 2005.

[Kam81] Hans Kamp. A theory of truth and semantic representation. In J. A. G. Groenendijk, T. M. V. Janssen, and M. B. J. Stokhof, editors, *Formal Methods in the Study of Language*, volume 1, pages 277–322. Mathematisch Centrum, Amsterdam, 1981.

[Kar15] Lauri Karttunen. From natural logic to natural reasoning. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 9041 of *Lecture Notes in Computer Science*, pages 295–309. Springer International Publishing, 2015.

[KR93] Hans Kamp and Uwe Reyle. *FROM DISCOURSE TO LOGIC-Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer Academic Publishers, 1993.

[KVGR11] Hans Kamp, Josef Van Genabith, and Uwe Reyle. Discourse representation theory. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic*, volume 15 of *Handbook of Philosophical Logic*, pages 125–394. Springer Netherlands, 2011.

[Mac09] Bill MacCartney. Natural language inference. Ph. D. dissertation, Stanford University, June 2009.

[Mit02] Ruslan Mitkov. *Anaphora Resolution*. Pearson Longman, London, 2002.

[MM07] Bill MacCartney and Christopher D. Manning. Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 193–200, Prague, June 2007. Association for Computational Linguistics.

[Mon70a] Richard Montague. English as a formal language. In Bruno Visentini et al., editors, *Linguaggi nella Società e nella Tecnica*, pages 189–224. Edizioni di Comunità, 1970.

[Mon70b] Richard Montague. Universal grammar. *Theoria*, 36(3):373–398, 1970.

[Mon73] Richard Montague. The proper treatment of quantification in ordinary English. In K. Jaakko J. Hintikka, Julius M.E. Moravcsik, and Patrick Suppes, editors, *Approaches to natural language*, pages 221–242. Dordrecht, 1973.

[Mos10] Lawrence S. Moss. *Logics for Natural Language Inference*. Unpublished, November 2010. Available at: http://www.indiana.edu/~iulg/moss/notes.pdf.

[Mos12] Lawrence S. Moss. The soundness of internalized polarity marking. *Studia Logica*, 100(4):683–704, 2012.

[MP14] Scott Martin and Carl Pollard. A dynamic categorial grammar. In Glyn Morrill, Reinhard Muskens, Rainer Osswald, and Frank Richter, editors, *Formal Grammar: 19th International Conference, FG 2014, Tübingen, Germany, August 16-17, 2014. Proceedings*, pages 138–154, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[MR12] Richard Moot and Christian Retoré. *The Logic of Categorial Grammars, A Deductive Account of Natural Language Syntax and Semantics*. Springer, 2012.

[NP12] Roberto Navigli and Simone Paolo Ponzetto. BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012.

[RNM05] Rajat Raina, Andrew Y. Ng, and Christopher D. Manning. Robust textual inference via learning and abductive reasoning. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 1099–1105. AAAI Press / The MIT Press, 2005.

[Sam11] Mark Sammons. Transformation and logic based approaches in rte. LSA Institute Workshop on Semantics for Textual Inference, University of Illinois at

Urbana-Champaign', 2011. Available at: http://cogcomp.cs.illinois.edu/member_pages/sammons/files/entailment-approaches.pdf.

[SB11] Mark Steedman and Jason Baldridge. Combinatory categorial grammar. In Robert Borsley and Kersti Borjars, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, pages 181–224. Wiley-Blackwell, 2011.

[SV91] Victor Sánchez-Valencia. Studies on natural logic and categorial grammar. Ph.D. thesis, Universiteit van Amsterdam, 1991.

[Uni10] WordNet Princeton University. Princeton university "about wordnet", 2010. http://wordnet.princeton.edu.

[vB86] Johan van Benthem. *Essays in Logical Semantics*, volume 29 of *Studies in Linguistics and Philosophy*. Reidel, Dordrecht, 1986.

[vB91] Johan van Benthem. *Language in Action: Categories, Lambdas, and Dynamic Logic*, volume 130 of *Studies in Logic*. Elsevier, Amsterdam, 1991.

[vB07] Johan van Benthem. A brief history of natural logic. Technical report, 2007. Institute for Logic, Language & Computation, University of Amsterdam. Available at: https://www.illc.uva.nl/Research/Publications/Reports/PP-2008-05.text.pdf.

[vE07] Jan van Eijck. Natural logic for natural language. In Balder ten Cate and Henk Zeevat, editors, *6th International Tbilisi Symposium on Logic, Language, and Computation Batumi, Georgia*, pages 216–230. Springer, 2007.