Statistical Deep Parsing for Spanish

Luis Chiruzzo

Instituto de Computación
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

A thesis submitted for the degree of

Doctor en Informática - PEDECIBA

Thesis supervisor: Dina Wonsever

December 2020

Jury Members

Laura Alonso Alemany (reviewer)
Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba, Argentina

Núria Bel
Professor, Senior Researcher
Universitat Pompeu Fabra, Spain

Héctor Cancela
Profesor Titular
Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

Brian Roark (reviewer)
Senior Staff Research Scientist
Google, United States of America

Paolo Rosso
Full Professor
Universitat Politècnica de València, Spain

Dina Wonsever (thesis supervisor)
Profesora Titular
Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

Abstract

This document presents the development of a statistical HPSG parser for Spanish. HPSG is a deep linguistic formalism that combines syntactic and semantic information in the same representation, and is capable of elegantly modeling many linguistic phenomena. Our research consists in the following steps: design of the HPSG grammar, construction of the corpus, implementation of the parsing algorithms, and evaluation of the parsers performance.

We created a simple yet powerful HPSG grammar for Spanish that models morphosyntactic information of words, syntactic combinatorial valence, and semantic argument structures in its lexical entries. The grammar uses thirteen very broad rules for attaching specifiers, complements, modifiers, clitics, relative clauses and punctuation symbols, and for modeling coordinations. In a simplification from standard HPSG, the only type of long range dependency we model is the relative clause that modifies a noun phrase, and we use semantic role labeling as our semantic representation.

We transformed the Spanish AnCora corpus using a semi-automatic process and analyzed it using our grammar implementation, creating a Spanish HPSG corpus of 517,237 words in 17,328 sentences (all of AnCora).

We implemented several statistical parsing algorithms and trained them over this corpus. The implemented strategies are: a bottom-up baseline using bilexical comparisons or a multilayer perceptron; a CKY approach that uses the results of a supertagger; and a top-down approach that encodes word sequences using a LSTM network.

We evaluated the performance of the implemented parsers and compared them with each other and against other existing Spanish parsers. Our LSTM top-down approach seems to be the best performing parser over our test data, obtaining the highest scores (compared to our strategies and also to external parsers) according to constituency metrics (87.57 unlabeled F1, 82.06 labeled F1), dependency metrics (91.32 UAS, 88.96 LAS), and SRL (87.68 unlabeled, 80.66 labeled), but we must take in consideration that the comparison against the external parsers might be noisy due to the post-processing we needed to do in order to adapt them to our format. We also defined a set of metrics to evaluate the identification of some particular language phenomena, and the LSTM top-down parser outperformed the baselines in almost all of these metrics as well.

Resumen

Este documento presenta el desarrollo de un parser HPSG estadístico para el español. HPSG es un formalismo lingüístico profundo que combina información sintáctica y semántica en sus representaciones, y es capaz de modelar elegantemente una buena cantidad de fenómenos lingüísticos. Nuestra investigación se compone de los siguiente pasos: diseño de la gramática HPSG, construcción del corpus, implementación de los algoritmos de parsing y evaluación de la performance de los parsers.

Diseñamos una gramática HPSG para el español simple y a la vez poderosa, que modela en sus entradas léxicas la información morfosintáctica de las palabras, la valencia combinatoria sintáctica y la estructura argumental semántica. La gramática utiliza trece reglas genéricas para adjuntar especificadores, complementos, clíticos, cláusulas relativas y símbolos de puntuación, y también para modelar coordinaciones. Como simplificación de la teoría HPSG estándar, el único tipo de dependencia de largo alcance que modelamos son las cláusulas relativas que modifican sintagmas nominales, y utilizamos etiquetado de roles semánticos como representación semántica.

Transformamos el corpus AnCora en español utilizando un proceso semi-automático y lo analizamos mediante nuestra implementación de la gramática, para crear un corpus HPSG en español de 517,237 palabras en 17,328 oraciones (todo el contenido de AnCora).

Implementamos varios algoritmos de parsing estadístico entrenados sobre este corpus. En particular, teníamos como objetivo probar enfoques basados en redes neuronales. Las estrategias implementadas son: una línea base bottom-up que utiliza comparaciones bi-léxicas o un perceptrón multicapa; un enfoque tipo CKY que utiliza los resultados de un supertagger; y un enfoque top-down que codifica las secuencias de palabras mediante redes tipo LSTM.

Evaluamos la performance de los parsers implementados y los comparamos entre sí y con un conjunto de parsers existententes para el español. Nuestro enfoque LSTM top-down parece ser el que tiene mejor desempeño para nuestro conjunto de test, obteniendo los mejores puntajes (comparado con nuestras estrategias y también con parsers externos) en cuanto a métricas de constituyentes (87.57 F1 no etiquetada, 82.06 F1 etiquetada), métricas de dependencias (91.32 UAS, 88.96 LAS), y SRL (87.68 no etiquetada, 80.66 etiquetada), pero debemos tener en cuenta que la comparación con parsers externos puede ser ruidosa debido al postprocesamiento realizado para adaptarlos a nuestro formato. También definimos un conjunto de métricas para evaluar la identificación de algunos fenómenos particulares del lenguaje, y el parser LSTM top-down obtuvo mejores resultados que las baselines para casi todas estas métricas.

Acknowledgements

I am deeply indebted to my thesis supervisor Dina Wonsever, this work would have not been possible without her thoughtful advice.

I would like to extend my sincere thanks to the NLP research group at Universidad de la República: Aiala Rosá for introducing me to the HPSG world, Mathias Etcheverry for his practical suggestions on taming neural networks, Diego Garat, Guillermo Moncecchi, Juanjo Prada, Santiago Castro and Santiago Góngora.

I must also thank Marisa Malcuori and Brenda Laca for their invaluable insight into Spanish linguistic peculiarities.

Special thanks to the HPSG community, in particular to Monsterrat Marimon, Yusuke Miyao and António Branco, for convincing me that this work was possible.

I would also like to thank the jury members, and in particular Brian Roark and Laura Alonso for reviewing this thesis.

Finally, I would like to thank my parents, family and friends for always being there. Many thanks to Alejandro Cholaquidis for his encouragement.

My deepest gratitude to Tatiana Rimbaud for her unconditional support, her infinite patience, and for never giving up on me.

This thesis is dedicated to Tatiana and Galatea.

Acojo en mi hogar
Palabras que he encontrado
abandonadas en mi palabrera
Examino cada jaula y allí
Ladrando vocales y consonantes
Encuentro sucios verbos
Que lloran después de ser
abandonados
Por un sujeto que un día fue su amo
Y de tan creído que era
Prescindió del predicado

Esta misma semana
Han encontrado a un par de adjetivos
transtornados
A tres adverbios muertos de frío
Y a otros tantos de la raza pronombre
Que sueñan en sus jaulas
Con ser la sombra de un niño

(...)

Pero todo es ley de vida

Como un día me dijo el poeta Halley
Si las palabras se atraen
Que se unan entre ellas
Y a brillar
Que son dos sílabas

El poeta Halley

Contents

Chapter 1

Introduction

Natural Language Processing is an interdisciplinary field that combines linguistics and artificial intelligence, its main intent is to create automatic methods that could understand or generate human language. This is undoubtedly a very ambitious objective, and we can say that even with the current state of the art, machines are not yet able to have a seamless conversation with humans. Nevertheless, the field advances relentlessly towards that goal, improving the performance on the tasks already defined and expanding the reach of the field to tasks that previously could not be solved.

One historical approach to Natural Language Processing [Jurafsky and Martin, 2009] proposes a possible set of linguistic knowledge that an NLP application should acquire, in increasing order of complexity:

- Phonetics and Phonology
- Morphology
- Syntax
- Semantics
- Pragmatics
- Discourse

Each of these items could be seen as steps in a pipeline that an NLP application would perform: starting from a raw sentence, apply certain analysis processes to extract linguistic information beginning from the earlier steps that collect simpler features, so that the later steps could use the features previously collected to build more complex representations. This pipeline is the backbone that enables classical approaches to higher order NLP tasks such as automatic summarization, question answering or machine translation. It is clear that in such a scenario, improvements in each of the steps lead to further improvements in downstream tasks. Although this pipeline approach is nowadays gradually

falling out of favor to more direct approaches like end-to-end systems that do not necessarily rely on having linguistic knowledge explicitly annotated to solve a problem, it is still widely used for many tasks, especially for languages with fewer linguistic resources than English. Besides seeking improvement in NLP tasks, the analysis of each one of the steps is in itself a very interesting problem that can shed light on important linguistic questions and, indirectly, give us insights on human language and the human mind itself.

Grammar

In the middle of the pipeline we find the stage of syntactic analysis. Syntax tries to model a language using a set of rules that govern how to build sentences. These sets of rules are known as **grammars**. Approaches to model natural language syntax have existed since antiquity (e.g. Pāṇini's Sanskrit grammar, the Aṣṭādhyāyī). Modern takes on the subject often try to incorporate linguistic and even cognitive approaches on both syntax and semantics. The process of taking a sentence and building a syntactic representation of it is known as syntactic analysis, and it is generally referred to as **parsing**. A system that performs this kind of analysis is referred to as a **parser**.

Parsing

Parser

There are several linguistic formalisms (grammars) that could be used to represent the syntax of a sentence. Some grammar formalisms are more shallow and just give surface information on the sentences and how their words interact, while others are deeper and often involve different types of syntactic and semantic information. One of these deep linguistic formalisms is Head-driven Phrase Structure Grammar (**HPSG**) [Pollard and Sag, 1994], a rich linguistic formalism that combines syntactic and semantic information in its analyses and is able to model many interesting linguistic phenomena. We will focus on this formalism in this work.

HPSG

Grammars and parsers can be developed with different intentions in mind. One approach is to build a grammar that is capable of correctly telling apart if a sentence is well-formed for a language or not. Such a grammar (and a parser that uses it) accepts some sequences of words as correct sentences of the language, and rejects others that might have errors. However, natural language is written by people, which on the one side are prone to make mistakes, and on the other hand are very creative so that new terms and expressions are coined everyday. Given this scenario, there is an approach to grammar development and parsing that does not strive to asses if a sentence is correct, but tries to infer what is the most likely syntactic structure a sequence of words could have. In this case, sentences with small errors (e.g. typos or minor grammatical mistakes) might still get a good enough analysis that could be used by downstream applications. These systems rely on statistical knowledge of the language, generally obtained from a **corpus**, a collection of sentences that is used as a representation of the language. Corpora used for parser development should be large, comprehensive and varied enough so as to capture the main characteristics of the language we want to model.

Corpus

1.1 Motivation

Historically, English has been the most explored language in NLP. The amount of data, both annotated and unannotated, that exists for English is not comparable to what exists for other languages. Similarly, the state of the art performance for different NLP tasks in English is often much higher than for other languages. In this regard, HPSG is no exception: there exist fast high performing parsers for English, but for other languages the research has in some ways lagged behind. In this work, we will try to further explore the adaptation of the HPSG deep linguistic formalism to Spanish.

This formalism has been explored for Spanish in the past: [Marimon et al., 2007] implemented a hand-crafted grammar for Spanish. This hand-crafted grammar might be a good way of describing the correctly formed sentences in the language, but in this work we take a rather different approach. Our aim is to build a statistical parser that could, given a sentence, return the most likely HPSG representation for that sentence. Because of this, our grammar will try to be as broad and generic as possible: we are not trying to tell apart if a sentence is well-formed or not, so the grammar should be able to capture a great number of sequences of words. In order to do this we need a corpus large enough to let us train a statistical parser, which we will create based on the information of an already established Spanish corpus.

It is our hope that this work could be able to narrow, even if very slightly, the existing gap between NLP capabilities for English and Spanish, and also that it could help foster the interest in developing and using resources for Spanish and for rich grammar formalisms.

1.2 Objectives

The objective of this work is to create a statistical HPSG parser for Spanish. In order to do this, the following milestones have to be completed:

- Define a HPSG grammar for Spanish that covers the language phenomena we want to address.
  We also want to make it flexible enough to be able to model many possible sentences, as we want to use it in the context of a statistical parser.

- Create a corpus of HPSG annotated sentences using that grammar.
  The parser would need to have a large collection of sentences in order to create good statistical models.

- Develop statistical parsing algorithms based on the information of the corpus.
  We will focus mainly on applying neural network techniques to the problem of HPSG parsing, which have not yet been widely explored for this task.

- Analyze the performance of the parsing algorithms compared to other established Spanish baselines.

1.3 Products of this thesis

During the course of this project we presented some partial results in different international venues. Sharing the early results of our research with the academic community gave us relevant feedback and helped us improve our work in numerous ways.

- *"Spanish HPSG Treebank based on the AnCora Corpus"* [Chiruzzo and Wonsever, 2018] was presented in the Eleventh International Conference on Language Resources and Evaluation (LREC-2018). The paper describes the HPSG grammar we use and the corpus we built by transforming the Spanish AnCora corpus, which will be described in detail in chapters 3 and 4.

- *"Building a supertagger for Spanish HPSG"* [Chiruzzo and Wonsever, 2019a] was published in volume 54 of the Computer Speech & Language journal. This paper describes some preliminary experiments we carried on supertagging with our HPSG grammar. The supertaggers developed for this paper were finally not used in the final parsers, but served as basis for the developments we will see in section 5.4.

- *"Syntactic Analysis and Semantic Role Labeling for Spanish using Neural Networks"* [Chiruzzo and Wonsever, 2019b] was presented as a poster in the 20th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing 2019). The paper presents some initial results of the parsing architecture described in section 5.5.

- *"Statistical Deep Parsing for Spanish Using Neural Networks"* [Chiruzzo and Wonsever, 2020] was presented as an oral presentation in the 16th International Conference on Parsing Technologies (IWPT 2020), a workshop of ACL 2020. The paper describes an initial comparison between two of the parsers we developed and some Spanish baselines. The description of the parsers and their results are expanded in this work in chapters 5 and 6.

It is also possible to test the parsing strategies we implemented in the web site `parsur.com`[1]. The site lets you parse sentences using the CKY and the LSTM top-down strategies and provides an interactive visualization of the resulting HPSG trees.

1.4 Document structure

The rest of this document is structured as follows:

Chapter 2 presents a series of concepts that will be used throughout the document. It starts by reviewing different types of grammar formalisms, in

[1] `http://parsur.com/`

particular HPSG, as well as parsing algorithms and linguistic resources. Then it describes important concepts on the field of machine learning and neural networks. Finally, it shows a review of the state of the art in parsing.

Chapter 3 presents the HPSG grammar adapted to Spanish we use in this project, introducing the feature structures, rules and principles.

Chapter 4 describes the transformation of the Spanish AnCora corpus into our HPSG format, the validation of the approach using our HPSG implementation and finally some statistics on the created corpus.

Chapter 5 introduces the concept of parsing in our grammar in a more formal way, together with the metrics used to evaluate the results of a parser. Then it describes the different parsing strategies we implemented and presents an initial comparison evaluating them against the development corpus.

Chapter 6 shows the actual evaluation of our approaches against the test corpus. We show the results of our parsers and compare them to other Spanish parsers in terms of global metrics, speed, and some particular metrics we defined for capturing interesting language phenomena.

Finally, chapter 7 presents the conclusions of our work and outlines some ideas that might be interesting to explore in the future.

Chapter 2

Background

This chapter presents important concepts about parsing and machine learning that will be used throughout this work. We begin by describing different grammar formalisms, parsing algorithms and resources, in particular the HPSG formalism we will be using. Then we present key ideas about machine learning, neural networks, word embeddings and neural parsing. We finish the chapter with a review of the state of the art in parsing both for English and Spanish.

2.1 Grammar formalisms

Within the field of Natural Language Processing, the task of parsing implies transforming a sentence in natural language to a representation in some formalism, usually generating tree structures or graphs. There are several formalisms that differ in the kind of information they represent, e.g. syntactic or semantic information, and the depth level of representation (surface parsing or deep parsing).

The two main families of syntactic representations are the ones based on constituents and the ones based on dependencies. The constituency based formalisms generally try to provide an analysis of a sentence in which the words are grouped in local units called phrases or constituents (some of them with linguistic foundations). On the other hand, the dependency based representation frames the analysis as a collection of relations between pairs of words (bi-lexical relations) named dependencies. In both cases, we say that the result of the parsing process of a sentence is a **parse tree** of the sentence, although there will be differences on the types of trees and information provided in each formalism.

Parse Tree

In this work we will focus mainly constituency based grammars and particularly on deep grammar formalisms whose representations include rich information and incorporate both syntactic and semantic information, but we will also sketch the main characteristics of several other formalisms.

2.1.1 Context Free Grammars

One of the first formalisms that became widely used is the Context Free Grammar (CFG) formalism. In this type of grammars, a parse tree is a tree where the leaves represent words and the inner nodes represent phrases containing syntactical information in the format specified by the grammar.

Formally, a CFG [Jurafsky and Martin, 2014] is a 4-tuple $< N, \Sigma, R, S >$ where:

Non-terminal

- N is a set of **non-terminal** symbols. These are used, for example, for modeling the different types of constituents.

Terminal

- Σ is a set of **terminal** symbols. These symbols are the words or other tokens used in the modeled language. Σ is the vocabulary.

Rule

- R is a set of **rules**. Each rule has the form $A \rightarrow \beta$, where $A \in N$ and $\beta \in (\Sigma \cup N)*$.
- S is a symbol that belongs to N and is called the start symbol. This is generally used to indicate which non-terminal represents the sentences of the grammar.

One very simple example of this type of grammars that could be used for modeling some Spanish sentences is shown below:

- $N = \{S,\ NP,\ VP,\ Det,\ Noun,\ Verb,\ Adj\}$[1]
- $\Sigma = \{la, gata, duerme, negra\}$
- $R = \{S \rightarrow NP\ VP,\ NP \rightarrow Det\ Noun,\ NP \rightarrow Det\ Noun\ Adj,$
  $VP \rightarrow Verb,$
  $Det \rightarrow la,\ Noun \rightarrow gata,\ Verb \rightarrow duerme,\ Adj \rightarrow negra\}$

Using this grammar we can analyze the following sentences:

(1) *La gata duerme / The cat sleeps*

(2) *La gata negra duerme / The black cat sleeps*

The trees shown in figure 2.1 are the corresponding syntactic analyses given by this grammar to the sample sentences 1 and 2. Each inner node in the tree corresponds to the application of a rule from R, where the parent node is the non-terminal and there is one daughter for each symbol on the right side of the rule. The leaves are the terminal symbols that correspond to the words of the sentence.

An important aspect of CFGs is that they can be used both for recognizing correct sentences in the language or for generating sentences. We say that a CFG recognizes a sentence if there exists a derivation (i.e. a sequence of applications of rules) that yields the corresponding sentence.

[1] In this example we are using the English names for the constituents, e.g., S is a sentence (*oración*), NP is a noun phrase (*sintagma nominal*) and VP is a verb phrase (*sintagma verbal*).

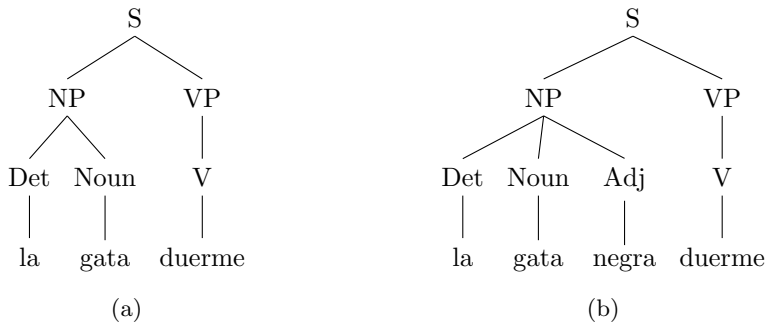


Figure 2.1: Syntactic trees using a CFG for samples 1 (a) and 2 (b).

In a CFG used for modeling a natural language, it is often the case that the non-terminal symbols that immediately precede the terminals represent the grammatical categories (either parts of speech or, as we will see later on, more complex categories that extend the parts of speech). This set of non-terminals that precede the terminals are also called **pre-terminals**.

Pre-terminal

Let us simplify the notation in the following way: Notice that, assuming all terminals and non-terminals are used in the grammar (so they appear in at least one rule) and that the start symbol is always S (sentence), we could fully determine a CFG just by knowing the rules set R. We can separate the set R in two subsets $R = R_g \cup R_v$, where R_g is the set of grammar rules, i.e., rules that operate over the non-terminals (including pre-terminals); while R_v is the set of rules that go from pre-terminals to terminals (from categories to words), defining the vocabulary. The previous grammar could be defined as follows:

- $R_g = \{S \rightarrow NP\ VP,\ NP \rightarrow Det\ Noun,\ NP \rightarrow Det\ Noun\ Adj,$ $VP \rightarrow Verb\}$
- $R_v = \{Det \rightarrow la,\ Noun \rightarrow gata,\ Verb \rightarrow duerme,\ Adj \rightarrow negra\}$

Using this notation we can start adding more vocabulary or more rules to the grammar in order to model more sentences of the language. Suppose we now want to model the following sentences:

(3) *El perro duerme / The dog sleeps*

(4) *El perro negro duerme / The black dog sleeps*

(5) *Las gatas duermen / The cats sleep*

We can do so by adding new items to the vocabulary R_v, leaving R_g unchanged:

- $R_v = \{Det \rightarrow la|las|el,\ Noun \rightarrow gata|gatas|perro,$ $Verb \rightarrow duerme|duermen,\ Adj \rightarrow negra|negro\}$

Now sentences 3, 4 and 5 are licensed by our grammar. But on the other hand, the grammar also generates other sequences of words that are not grammatical sentences, such as the following:

(6) * *El gata duerme*

(7) * *El perro negra duerme*

(8) * *La gata duermen*

Agreement

Natural languages such as English and Spanish present a phenomenon called **agreement** which means that within constituents some of the words must have matching morphological characteristics (they must *agree*). For example, in Spanish the determiner and the adjectives in a noun phrase must agree in number and gender to the noun; and also the subject of a sentence must agree in number to the main verb of the sentence. We can see that sentences 6 and 7 violate the noun phrase agreement principle, while 8 violates the subject-verb agreement principle for Spanish.

One way of fixing this would be creating new non-terminals that help us determine the behavior of each word further. For example, we could indicate the number and gender of each word using F (female), M (male), S (singular) or P (plural) as suffixes for the pre-terminals:

- $R_v = \{Det_{FS} \to la,\ Det_{FP} \to las,\ Det_{MS} \to el,$
  $Noun_{FS} \to gata,\ Noun_{FP} \to gatas,\ Noun_{MS} \to perro,$
  $Verb_S \to duerme,\ Verb_P \to duermen,$
  $Adj_{FS} \to negra,\ Adj_{MS} \to negro\}$

This also means we have to change the non-terminals that use these pre-terminals:

- $R_g = \{S \to NP_S\ VP_S,\ S \to NP_P\ VP_P,$
  $NP_S \to Det_{FS}\ Noun_{FS},\ NP_P \to Det_{FP}\ Noun_{FP},$
  $NP_S \to Det_{MS}\ Noun_{MS},\ NP_P \to Det_{MP}\ Noun_{MP},$
  $NP_S \to Det_{FS}\ Noun_{FS}\ Adj_{FS},\ NP_P \to Det_{FP}\ Noun_{FP}\ Adj_{FP},$
  $NP_S \to Det_{MS}\ Noun_{MS}\ Adj_{MS},\ NP_P \to Det_{MP}\ Noun_{MP}\ Adj_{MP},$
  $VP_S \to Verb_S,\ VP_P \to Verb_P\}$

Previously we had only one way of writing a sentence S or a verb phrase VP, now there are two of each because we must define S or P versions. Even worse, previously we had two ways of writing a noun phrase NP, with or without an adjective, but not we must define four times more rules, because we have FS, FP, MS or MP versions of this constituent.

So far the sentences we can model with out grammar are very simple, once we start adding transitive verbs, ditransitive verbs, subject or object control structures, or many other language phenomena that are interesting to model, the number of rules starts to grow exponentially. This means that only using CFGs for natural language becomes quickly an intractable problem.

Nonetheless, CFG grammars might still be used as a simple way to model a natural language if we only want to capture some of the phenomena. These problems could be solved more easily, as we will see later on, using other types of grammars that work with feature structures and can define constraints between the features.

Let us see two more examples and how they can be modeled using a CFG:

(9) *La gata come el pescado / The cat eats the fish*

(10) *Juan da pescado a la gata / John gives fish to the cat*

So far the only verb we modeled (*"duerme" / "sleeps"*) had only one argument, which is the subject. Verbs whose only argument is the subject are called *Intransitive Verb* **intransitive verbs**, but there are other types of verbs that could accept other types of arguments. Sentence 9 uses the verb *"come"* (*"eats"*), which is a *Transitive Verb* **transitive verb**: a verb that requires a subject and a direct object. On the other *Ditransitive Verb* hand, sentence 10 uses the verb *"da"* (*"gives"*), which is a **ditransitive verb**, requiring a subject, a direct object and an indirect object as arguments. These are not the only types of verbs, as there are verbs that accept only some kinds of complements, or none of them, or impose different properties over the types of arguments. The capacity of a verb to select the types of arguments it can be *Subcategorization* combined with is called the **subcategorization** of the verb, and modeling it is an important problem when developing a grammar.

One way of modeling these two new sentences in our grammar would be to add particular rules for the transitive and intransitive verbs (besides the rules for modeling the rest of the structures and terminals). For example we could extend the ways we build VPs like the following:

- $VP \rightarrow Verb$
- $VP \rightarrow Verb\ NP$
- $VP \rightarrow Verb\ NP\ PP$

These new rules would allow us to analyze all the previous sentences plus the new sentences, but adding new rules for each possible subcategorization might make the grammar grow in an uncontrollable way and lead to a combinatorial explosion like the one we saw for the agreement case. Another way of representing these same three options would be the following:

- $VP \rightarrow Verb$
- $VP \rightarrow VP\ NP$
- $VP \rightarrow VP\ PP$

Binarization In this case we have a **binarized** version of the previous rules, which is conceptually simpler. We can easily see that any sentence accepted by the previous set of rules would also be accepted by these rules. The corresponding parse trees might be a little deeper. However, it is also true that these new

rules license more sentences than the previous ones, as one can arbitrarily add more NPs or PPs to the right of a verb phrase. This might not be a bad thing, perhaps we want the grammar to be able to model any combination of complements on the right of a verb, this will depend on the language. We can see there could be a trade-off between the simplicity of the grammar and its expressive power.

Binarization can also be applied to other rules, like $NP \rightarrow Det\ Noun\ Adj$ in order to make them simpler or more flexible (you could add any number of adjectives to a noun phrase), and it is mandatory for some parsing algorithms (see section 2.2.1).

2.1.2 Probabilistic Context Free Grammars

Consider the following sample sentences 11 and 12:

(11) *La gata de Juan come pescado / John's cat eats fish*

(12) *La gata come pescado de noche / The cat eats fish in the night*

We could change and extend the grammar we have been using in the following way to analyze those sentences:

- $R_v = \{Det \rightarrow la,\ Noun \rightarrow gata|pescado|Juan|noche,$
  $Verb \rightarrow duerme|come,\ Adj \rightarrow negra,\ Prep \rightarrow de\}$
- $R_g = \{S \rightarrow NP\ VP,$
  $NP \rightarrow Nom,\ NP \rightarrow Det\ Nom,$
  $Nom \rightarrow Noun,\ Nom \rightarrow Noun\ Adj,\ Nom \rightarrow Noun\ PP,$
  $VP \rightarrow Verb,\ VP \rightarrow Verb\ NP,\ VP \rightarrow Verb\ NP\ PP$
  $PP \rightarrow Prep\ NP\}$

Figure 2.2 shows the analysis for sample 11, but for sample 12 the grammar allows two possible analyses: either the prepositional phrase *"de noche"* (*"in the night"*) is modifying the verb *"come"* (*"eats"*) or it is modifying the noun *"pescado"* (*"fish"*). A speaker of the language would have no problem in identifying that the appropriate analysis is the first one. However, there is *a priori* no way of telling that from the grammar, so we say the sentence is **ambiguous** with respect to this grammar.

Ambiguity

This particular type of ambiguity is very well known in NLP as a pervasive problem that is very hard to solve with automatic techniques. Within the NLP literature this is called the **PP-attachment** (prepositional phrase attachment) problem.

PP-attachment

One way of dealing with this ambiguity is including some statistical information in the grammar so that we can choose the most likely analysis from the set of possible trees. The Probabilistic Context Free Grammars are a kind of grammar very similar to CFGs but incorporating this statistical information. The main difference between a PCFG and a CFG is that it has a probability defined for each rule. So we change the previous definition of the set R in section 2.1.1 to the following:

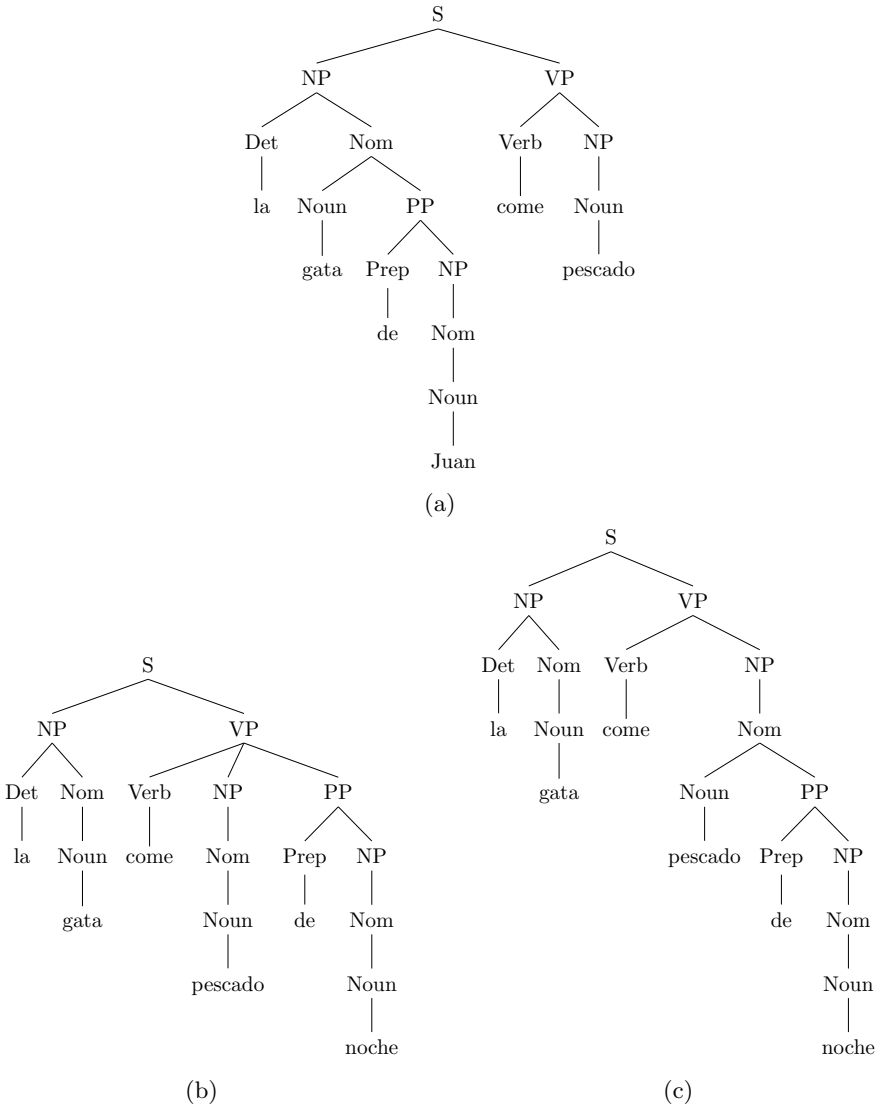


Figure 2.2: Syntactic tree using CFG for sample 11 (a) and two possible analysis for sample 12 (b) and (c).

- R is a set of rules. Each rule has the form $A \rightarrow \beta$ with an associated probability $P(A \rightarrow \beta)$, where:
  - $A \in N$
  - $\beta \in (\Sigma \cup N)*$
  - $\sum_{\beta} P(A \rightarrow \beta) = 1$

Rule	Prob	Rule	Prob
$S \rightarrow NP\ VP$	1.0	$Det \rightarrow la$	1.0
$NP \rightarrow Nom$	0.5	$Noun \rightarrow gata$	0.25
$NP \rightarrow Det\ Nom$	0.5	$Noun \rightarrow pescado$	0.25
$Nom \rightarrow Noun$	0.33	$Noun \rightarrow Juan$	0.25
$Nom \rightarrow Noun\ Adj$	0.33	$Noun \rightarrow noche$	0.25
$Nom \rightarrow Noun\ PP$	0.33	$Verb \rightarrow duerme$	0.5
$VP \rightarrow Verb$	0.25	$Verb \rightarrow come$	0.5
$VP \rightarrow Verb\ NP$	0.25	$Adj \rightarrow negra$	1.0
$VP \rightarrow Verb\ NP\ PP$	0.5	$Prep \rightarrow de$	1.0
$PP \rightarrow Prep\ NP$	1.0		

Table 2.1: Possible probability assignment for the grammar rules.

As in this formalism we assume the independence between branches of a tree, the probability of a particular tree for a sentence can be calculated as the product of the probabilities of all the rules used in the tree. In a consistent grammar, the probabilities of all the possible sentences generated by the grammar will add up to 1.

Continuing with the example, we need to assign probabilities for all the rules, both the vocabulary rules and the grammar rules. Table 2.1 shows a possible assignment of probabilities for each rule.

Using these probabilities we might calculate the probabilities for the trees 2.2b and 2.2c in the following way:

$$\begin{aligned}
P(tree_{2.2b}) = P(S \rightarrow NP\ VP)P(NP \rightarrow Det\ Nom)P(Nom \rightarrow Noun) \\
P(VP \rightarrow Verb\ NP\ PP)P(NP \rightarrow Nom)P(Nom \rightarrow Noun) \\
P(PP \rightarrow Prep\ NP)P(NP \rightarrow Nom)P(Nom \rightarrow Noun) \\
P(Det \rightarrow la)P(Noun \rightarrow gata)P(Verb \rightarrow come) \\
P(Noun \rightarrow pescado)P(Prep \rightarrow de)P(Noun \rightarrow noche) \\
= 1 \cdot 0.5 \cdot 0.33 \cdot 0.5 \cdot 0.5 \cdot 0.33 \cdot 1 \cdot 0.5 \cdot 0.33 \cdot 1 \cdot 0.25 \cdot 0.5 \cdot 0.25 \cdot 1 \cdot 0.25 \\
= 0.000017547 \\
P(tree_{2.2c}) = P(S \rightarrow NP\ VP)P(NP \rightarrow Det\ Nom)P(Nom \rightarrow Noun) \\
P(VP \rightarrow V\ NP)P(NP \rightarrow Nom)P(Nom \rightarrow Noun) \\
P(PP \rightarrow Prep\ NP)P(NP \rightarrow Nom)P(Nom \rightarrow Noun\ PP) \\
P(Det \rightarrow la)P(Noun \rightarrow gata)P(Verb \rightarrow come) \\
P(Noun \rightarrow pescado)P(Prep \rightarrow de)P(Noun \rightarrow noche) \\
= 1 \cdot 0.5 \cdot 0.33 \cdot 0.25 \cdot 0.5 \cdot 0.33 \cdot 1 \cdot 0.5 \cdot 0.33 \cdot 1 \cdot 0.25 \cdot 0.5 \cdot 0.25 \cdot 1 \cdot 0.25 \\
= 0.000008774
\end{aligned}$$

As we can see, the probability for tree 2.2b is higher than the one for tree 2.2c, which for our example corresponds to the intuition that the preferred tree

is 2.2b. PCFGs can be used to disambiguate parse trees obtained by a CFG. The catch in this example is that we defined the probabilities so that the probability of the verb phrase rule used in the first tree ($VP \rightarrow Verb\ NP\ PP$) is higher than the one used in the second tree ($VP \rightarrow Verb\ NP$).

Of course, this is a very simple example using arbitrary probabilities. When the number of rules and words starts to grow, assigning these probabilities by hand becomes unfeasible. The easiest way to assign probabilities to a PCFG is to extract them from a **treebank**, i.e., a corpus of appropriately labeled trees. We will discuss treebanks in section 2.4.

Treebank

The standard PCFGs described so far are not exempt from problems. Consider the following samples:

(13) *María come arroz con tenedor / Mary eats rice with a fork*

(14) *María come arroz con leche / Mary eats rice pudding*

Both examples have the same ambiguity: the prepositional phrases *"con tenedor"* or *"con leche"* might be attached to the verb *"come"* or to the noun *"arroz"*. The particularity of these examples is that the ambiguity cannot be solved using the rules that correspond to the VP as in the previous example, because the correct rule to use in both examples is different. Example 13 should use rule $VP \rightarrow Verb\ NP\ PP$ because *"con tenedor"* describes an instrument associated to the action *"come"*. However, example 14 should use rule $VP \rightarrow Verb\ NP$ and later on attach *"con leche"* to the noun *"arroz"* because it represents an ingredient that is modifying the food. So the trick of changing only the probability of the VP rule will not be enough in this case. We could instead start tinkering with the probabilities of the nouns, but we will probably run into the same situation when we want to use these nouns in other contexts.

This situation points to a larger problem in this type of PCFGs: as the probabilities are assigned to rules composed of non-terminals, regardless of the lexical entries they will be materialized on, they fail to capture the interactions that might happen between these lexical entries. In this case, the disambiguation should take into account that *"tenedor"* could often be used an instrument for *"comer"*, while *"arroz con leche"* is an expression that denotes a type of dish, and the relationship between these words and other verbs of nouns might not be as strong.

Lexicalized grammar

The way to deal with this in a more expressive way is by using **lexicalized grammars**, i.e., grammars that pay more attention into modeling the words (lexical entries) and the interactions between them. There are extensions to PCFGs that account for lexicalized items, such as the one described in [Collins, 1997], but in this work we will focus on another type of lexicalized grammar that include specific information into the lexical entries so as to guide the parsing process, as we will see in the following sections.

2.1.3 Dependencies

Dependency

Head
Dependent

Root

Dependency tree

Dependency grammars are not based on the idea of identifying constituents in a sentence, but only identifying relations (called **dependencies**) between the words of the sentence. The dependencies are always determined between pairs of words, denominated a **head** and a **dependent**, and thus they are called bi-lexical dependencies. Every word in a sentence has a dependency to another word (its head), and this dependency is noted as an arc that is labeled with the name of the relation between the words (e.g. subject, direct object, specifier). There is one word in the sentence that does not depend on any other word, which is called the **root** of the sentence.

The set of dependencies in a sentence form a directed acyclic graph which is denominated the **dependency tree**. Formally, a dependency tree [Jurafsky and Martin, 2014] is a structure $G = (V, A)$ where the set or vertices V corresponds to the words of the sentence, and the set of arcs A corresponds to the bi-lexical dependencies in the sentence. These arcs are often labeled with a relation name. A dependency grammar is defined with the set of labels D that could be used to label the arcs of a tree.

For example, let us assume that we have a grammar with the following set of dependencies:

$$D = \{\texttt{subject}, \texttt{direct\_object}, \texttt{specifier}, \texttt{modifier}\}$$

Sentences 1 through 5 and also 9 and 10 can be modeled in this grammar. Figure 2.3 shows the dependency trees corresponding to samples 1 (*"La gata duerme"*) and 9 (*"La gata come el pescado"*).

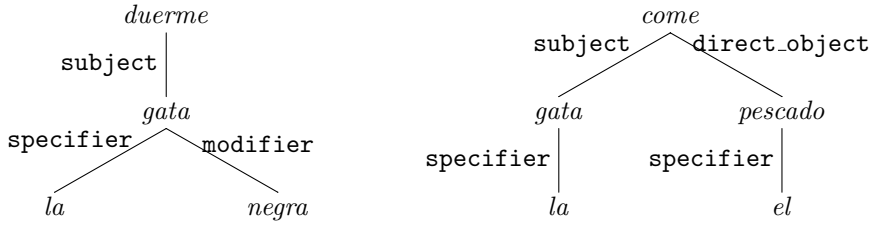


Figure 2.3: Syntactic trees using dependencies for samples 2 (a) and 9 (b).

However, notice that samples 6 through 8 could also be drawn as graphs in this dependency format, even if they are ungrammatical. We usually talk about dependency grammars, but this formalism generally does not provide a way of accepting or rejecting a sequence as a valid sentence, as CFG and other grammars do.

In this formalism the relations are established directly between pairs of words and potentially any sequence can be transformed into a dependency graph. In this case, the task of deciding if a sequence is properly labeled as a sentence becomes, instead, the problem of finding the most likely assignment of arcs

and labels to the set of nodes, a fundamentally statistical problem. The use of treebanks that indicate which associations are more frequent than others is crucial for this.

Different implementations of dependency grammars also vary in how they define headness, i.e. what word should be considered the head of a linguistic structure. Remember these grammars have no notion of constituents, but often dependency treebanks are built from the transformation of corpora annotated in a constituency grammar like CFG. For these conversions, it is important to define which word inside a constituent should be indicated as the head, and how the other words relate to it. For example, some head-finding rules [Collins, 2003] for the Penn Treebank (see section 2.4.1) consider that the head of a prepositional phrase should be the preposition. However, when building the dependency trees using Universal Dependencies (see section 2.4.3), we would choose the head to be the content word instead of the function word.

Projectivity

Another important concept in dependency formalisms is the notion of **projectivity**. A dependency tree is said to be projective if it is possible to draw it down without crossing arcs, or more formally: if there is an arc from head i to dependent j, there should be a path from i to k for every k between i and j. If we take the trees shown in figure 2.3 and draw them so that the words follow the same order as in the sentence, the resulting trees would be projective, but consider the following sentence:

(15) *Vi un perro ayer que era negro / I saw a dog yesterday that was black*

A dependency tree for sentence 15 is shown in figure 2.4. Notice that the `mod` arc from 1 to 4 crosses paths with the `rel` arc from 3 to 5, thus this is a non-projective dependency tree. Natural languages contain some of these non-projective structures that are not easy to model. For example, CFG grammars always generate trees that are projective when converted to dependencies, so a corpus made by transforming CFG-annotated trees will contain only projective trees. Furthermore, some dependency parsing algorithms can only find projective trees.

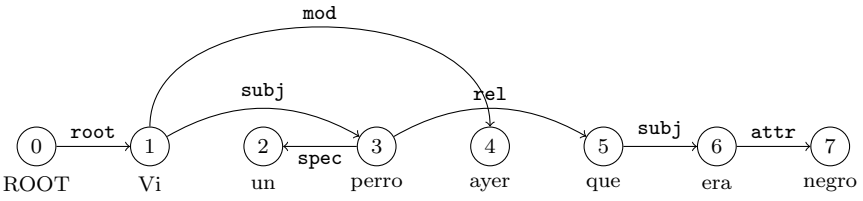


Figure 2.4: Non-project dependency analysis for the sentence *"Vi un perro ayer que era negro"* (*"I saw a dog yesterday that was black"*).

2.1.4 Head-driven Phrase Structure Grammars

We have seen so far grammars like CFG that focus on modeling the constituents of a sentence, and lexicalized dependency grammars that focus on modeling the relations between words without considering constituents. There exist other types of grammars that, while strongly lexicalized, still focus on modeling the constituency structure of the sentence. These grammar formalisms are generally called deep linguistic formalisms and they can deal with a wide range of linguistic phenomena and also could incorporate both syntactic and semantic information into their parse trees.

Deep formalisms are strongly lexicalized, which means much of the complexity of the grammars is encoded in the description of its lexical entries. Because of this, these formalisms often have very few rules and the rules can generally be applied in a broad range of scenarios, but only when they are licensed by the lexical entries. Examples of these formalisms include Combinatory Categorial Grammar [Steedman, 1996], Tree Adjoining Grammar [Joshi, 1985], Lexical Functional Grammar [Dalrymple, 2001] and Head-driven Phrase Structure Grammar [Pollard and Sag, 1994]. These deep grammars differ in the way they describe the lexical entries, how they can be combined to form greater units, and the shape of the structure that represents a parse tree in each grammar. In this work we will focus on Head-driven Phrase Structure Grammars, see appendix A for a description of other types of deep grammars.

Head-driven Phrase Structure Grammar (HPSG) is a strongly lexicalized grammar formalism based on feature structures with a unification operation.

Feature structure

Each lexical entry is represented with a **feature structure**, i.e., an association of features and values that describes the behavior of the word. Feature structures can be combined using **unification**: a process by which the features of two feature structures are merged forming a new one containing the features of both, and goes on recursively unifying the values when the same feature is found in both structures.

Unification

Typed feature structure

Standard HPSG uses **typed feature structures** (TFS) [Carpenter, 1992], which include an ontology-based type hierarchy definition for all the possible features and values present in the grammar.

The rules of the grammar and eventually the whole parse trees are also represented as feature structures. Every rule defines one of its daughters as the syntactic head of the rule, and there are a series of **principles** that indicate the way the daughters features are percolated to the parent structure. The grammar is head-driven because the principles indicate that, unless explicitly specified, the parent structure will inherit the features of the daughter marked as head.

Principle

Suppose we want to analyze sample sentence 2 *"La gata negra duerme"* using an HPSG grammar. We first need to define the lexical entries, as shown in figure 2.5. Notice that lexical entries are represented by feature structures (drawn as attribute-value matrices) and each one of them includes a `HEAD` feature. This feature defines the part of speech of the lexical entry together with its morphological attributes. The `VAL` feature includes the combinatorial in-

formation for the words, i.e., how they could be combined with other words. Words can indicate whether they are expecting a specifier (feature `SPEC`) or some complements, or if they are expecting to modify another word (feature `MOD`). The expressions written between angled brackets < and > indicate a list of values, and each one of the values can declare constraints on the types of elements that can be selected. For example, the adjective *"negra"* indicates its modified element must have the part of speech noun.

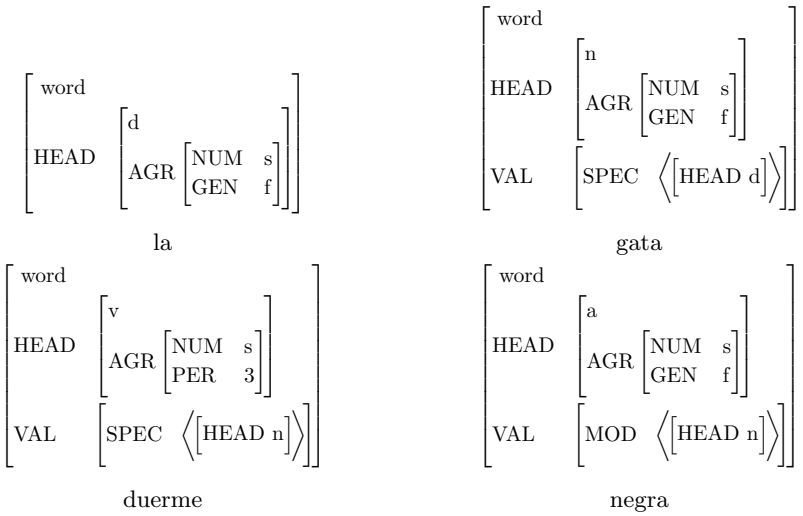


Figure 2.5: HPSG lexical entries for *"La gata negra duerme"*.

Now we need to define some rules for our grammar. For our simple example we will only need two rules: a rule for attaching a modifier to the right of a head (`head_mod`), and a rule for attaching a specifier to the left of a head (`spec_head`). Figure 2.6 shows these rules.

We will base our discussion on the HPSG version presented in [Sag et al., 2003] which, unlike the original [Pollard and Sag, 1994], uses the same specifier rule for indicating a determiner-noun construction and a subject-verb construction. That is why we only need the `spec_head` rule for representing both constructions.

Because much of the combinatorial information is encoded in the lexical entries and not in the rules, there tend to be very few rules in an HPSG grammar compared to a CFG. These are generally very broad rules, inspired mainly in X' theory ([Chomsky, 1970], [Chomsky, 1995]), for example describing the way to attach a specifier, a modifier or a complement to a head.

Figure 2.6 shows the description of the two rules `head_mod` and `spec_head`. In a rule definition, we show the daughters of the rule on the left side of the arrow, and the parent on the right side[2], which is the expression that results of combining the daughters according to this rule. Both `spec_head` and `head_mod`

[2]Compare this with the CFG rules presented in section 2.1.1, which were written the other

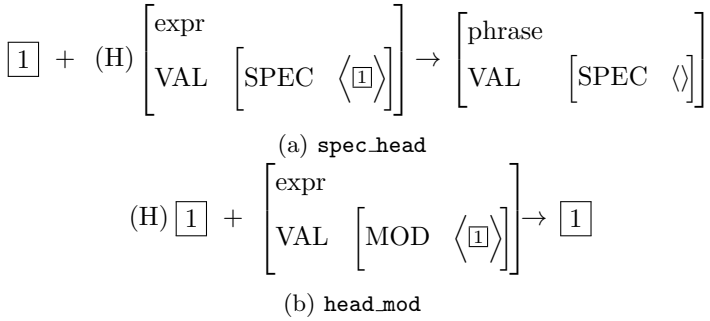


Figure 2.6: HPSG rules for attaching a specifier to the left of a head, and for attaching a modifier to the right of a head.

are binary rules: they take exactly two expressions as daughters. Notice that the daughter that is the syntactic head in each rule is signaled with a "(H)" mark.

Figure 2.6a shows that `spec_head` takes an expression on the left, and another expression on its right that declares to be expecting a specifier, the result of the application of the rule is another expression that will not expect a specifier (hence the empty list for the `SPEC` feature). An important characteristic of the formalism is the use of **coindexation**: marking two or more elements in a structure as being the same (they are unified and point to the same substructure). This means that in the final tree the head expression will have a pointer to the expression marked as specifier.

Coindexation

In figure 2.6b we see that `mod_head` takes an element on the left that will act as head of the construction, and combines it with an expression on its right that declares to be expecting another element to modify (the feature `MOD`). The result will share the same properties as the head, but also after applying the rule the `MOD` feature of the right daughter will be coindexed with the left daughter, thus in the tree the modifier element will have a pointer to its corresponding modified head.

Notice a difference between the `SPEC` and `MOD` features. `SPEC` is a feature belonging to a head that declares to be expecting some object as a specifier. We call this an **endocentric feature**, i.e. a feature from a head that points to a dependent structure outside the head. Contrast this with the `MOD` feature, that is defined by the modified element and points back to the head. This is an example of an **exocentric feature**, i.e. a feature from a dependent that points to the head of the construction.

Endocentric feature

Exocentric feature

Figure 2.7 shows the derivation steps for analyzing sentence 2 *"la gata negra duerme"* using our small HPSG grammar. The first step, shown in figure 2.7a, is combining *"gata"* and *"negra"* using the `head_mod` rule. Notice that this leaves the head *"gata"* coindexed with the `MOD` feature of *"negra"*, and the parent of

way round with the parent on the left. This is only a matter of notation style, as both types of definitions are analogous.

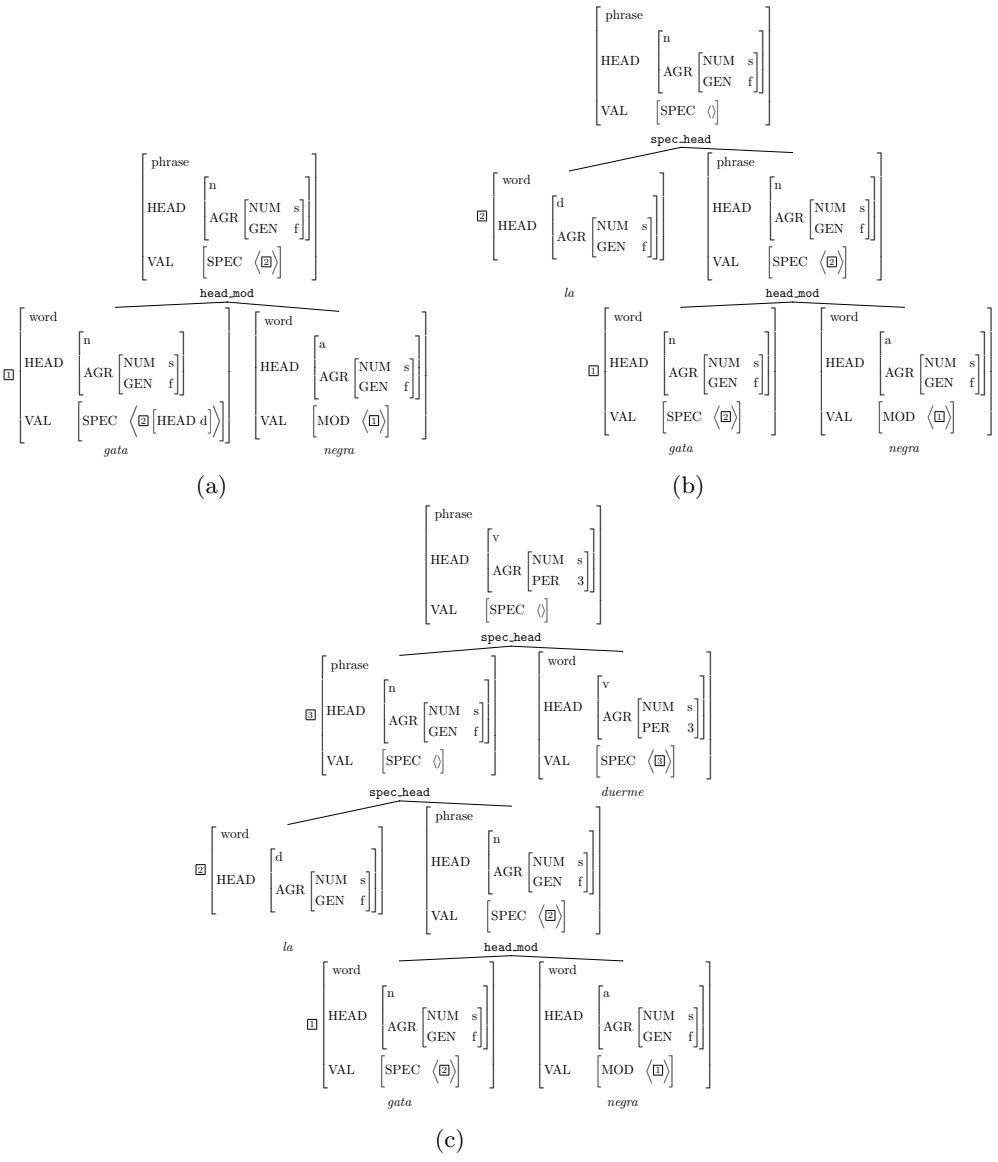


Figure 2.7: HPSG derivation for *"La gata negra duerme"*.

the structure contains the same features as the head daughter, even expecting the same `SPEC` as it is coindexed with the daughter's feature.

The second step, shown in figure 2.7b, combines the result of the first step with the determiner *"la"* using the `spec_head` rule. The result will have an empty value for `SPEC`, which means the specifier that the daughter was expecting has been satisfied. When this happens, we say that this valence feature has been

Saturated feature

saturated in this construction. The tree now indicates the word *"la"*, and the `SPEC` features of both *"gata"* and its parent expression as coindexed, so the head *"gata"* effectively has a pointer to its specifier *"la"*.

Figure 2.7c shows the last step of this derivation, the result of combining the previous structure *"la gata negra"* with the verb *"duerme"* using the `spec_head` rule. After this rule application, the `SPEC` feature of the verb gets saturated, and *"duerme"* gets a pointer to its subject *"la gata duerme"*. The parent expression is a verb with all its valence features saturated, which is the way a sentence is defined in this formalism.

Let us revisit the problem of modeling *agreement* described in section 2.1.1. Suppose we want to analyze the ungrammatical sentence 6 *"el gata duerme"*. We would need the following lexical entry for modeling the word *"el"*:

$$\begin{bmatrix} \text{word} & \\ \text{HEAD} & \begin{bmatrix} \text{d} & \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{s} \\ \text{GEN} & \text{m} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Agreement principle

It is essentially the same as the one for word *"la"*, but changing the gender agreement feature to `m` (male). HPSG defines the **agreement principle** that states that, when applying the `spec_head` rule, the `AGR` features of the head and the dependent should unify. It we try to apply the `spec_head` rule to *"el"* and *"gata"*, the `AGR` features of both lexical entries would fail to unify (`GEN` is `m` for *"el"* but `f` for *"gata"*), thus the rule could not be properly applied and the derivation of the tree would fail.

The same thing would happen if we try to analyze sentence 8 *"la gata duermen"*. In this case the `NUM` feature corresponding to *"duermen"* would be `p` (plural), while it is `s` (singular) for *"gata"*.

Although the original theory proposed for English does not include it, we could extend the agreement principle and apply it to the `head_mod` rule as well in order to capture the noun-adjective agreement existing in Spanish. With this extension, sentence 7 *"el perro negra duerme"* would also be ruled out as the `head_mod` rule could not be applied to *"perro"* and *"negra"*.

HPSG grammars are also very good at modeling the subcategorization of verbs, as each verb can be modeled with a lexical entry that specifies exactly the types of complements and subject it expects and, as we will see, it can also link this information with a semantic representation.

2.1.5 Semantic representations

Besides the syntactic representations we have seen so far, there exists particular interest in finding representations of a sentence that are more closely related to the meaning of the sentence rather than its surface realization. The study of these representation is part of the field of computational semantics.

First-order Logic

One of the first ideas for representing the meaning of a sentence is using first-order logic. First-order logic is a flexible tool that allows to create a meaning representation of many situations by defining the set of participants and relations between them. For example, suppose we want to represent the meaning of the following sentence:

(16) *Juan persiguió un ratón / John chased a mouse*

One possible representation for this sentence in first-order logic could be the following:

$$\exists x \cdot raton(x) \wedge persiguio(\texttt{juan}, x)$$

To make this semantic representation, we must define a few elements of the reality we are trying to model. In our example we would treat proper nouns (like *"Juan"*) as constants in the domain (`juan`), while common nouns (like *"ratón"*) could be seen as unary predicates ($raton(x)$) so they can take variable values depending on the intended meaning. The actions or states in a sentence would also be represented as predicates. For example a transitive verb like *"persiguió"* would correspond to a binary predicate $persiguio(x, y)$.

This idea was explored by [Montague, 1970] for English, defining a semantic expression for each word and grammar rule in the language that consists in a first-order logic term wrapped into a lambda expression. This allowed for the combination of words and expressions into more complex terms, finally reaching the full first-order logic representation of the sentence.

Later on this basic first-order logic representation was made more expressive by the inclusion of event representations in the form of event variables ([Davidson, 1967; Parsons, 1990]). This means that instead of representing a situation like *"persiguió"* as a binary predicate $persiguio(x, y)$, we would create an event for the "situation of chasing" $perseguir(x)$, and other predicates for indicating the different participants in the situation and other features like the time of the action. In our example:

$$\exists e \exists x \cdot perseguir(e) \wedge raton(x) \wedge perseguidor(e, \texttt{juan}) \wedge perseguido(e, x) \wedge tiempo\_pasado(e)$$

This can be interpreted as: there is a *chasing* situation e ($perseguir(e)$) in which the *chaser* is *"Juan"* ($perseguidor(e, \texttt{juan})$), the *chased thing* is the mouse ($perseguido(x)$), and the action occurs in the past ($tiempo\_pasado(e)$). The advantage of this representation over the previous one is that it is flexible

with respect to the different arguments and modifiers that could be used to change the modeled situation, as they could be added as different terms that predicate over the event variable e.

Compositional semantics

An important aspect associated with semantic representations like first-order logic and others is the notion of **compositional semantics**: the meaning representation of a larger unit can be built by combining the meaning representations of its parts. This fits nicely with the notion of compositionality of the language and trying to link the semantics representation to one of the syntactic formalisms, for example the lambda expressions used for building first-order logic predicates can be attached to terminals and non-terminals in a CFG grammar so as to let the syntax guide the semantic analysis. This is not always possible, however, because on many occasions the language presents **non-compositional semantics**, i.e. when the meaning of an expression cannot be built up from the meaning of its parts. This happens, for example, in idioms like *"dar una mano"* (*"lend a hand"*) which literally means *"ayudar"* (*"to help"*), but its meaning cannot be built up from the literal meaning of the words.

Non-compositional semantics

Semantic Roles

The notion outlined in the previous section about modeling the meaning of a sentence as a set of events with their participants and establishing the relations between them leads us to another popular format for representing meaning: the use of semantic roles. **Semantic roles** are a way of describing the relation that different constituents of a sentence might have with respect to a predicate (mainly verbs, but there could be other types of predicates). For example in sentence 16, we could say that *"Juan"* is the **agent** (i.e. the participant that causes the action) of *"perseguir"*, while *"un ratón"* is the **theme** (i.e. the participant that suffers the effects of the action). There is a rather large set of standard roles that could be used, including the aforementioned `agent` and `theme`, but also `instrument`, `force`, `source`, `destination`, etc.

Semantic Role

Agent

Theme

Different theories might work with different sets of possible semantic roles, so there have been some attempts to unify these theories into a consistent set of categories. One of these approaches is PropBank (short for Proposition Bank) [Kingsbury and Palmer, 2002], a project for annotating all the predicate-argument structures found in the Penn Treebank (see section 2.4.1). In order to create this corpus, they had to build a unified annotation framework for labeling the different roles in a consistent way. This annotation style [Bonial et al., 2010] has been extended to many languages other than English and has become a *de facto* standard in semantic roles annotation.

In PropBank notation, they define the semantic roles associated to a predicate using numbered labels like `arg0` though `arg4` and some special labels like `argM` (although some implementations might use a few other labels). The meanings of some labels are fixed, but others might change from verb to verb:

- The label `arg0` is called the *proto-agent*, and is reserved for denoting the predicate argument that causes the action, which in other theories might

involve the `agent`, `force`, or other types of roles.

- The label `arg1` is called the *proto-patient*, and is reserved for denoting the predicate argument that is affected by the action, which in other theories might involver the `theme`, `patient`, or other types of roles.
- The rest of the labels depend on the predicate. In general `arg2` will be used for roles like `instrument`, `benefactive` or `attribute`, while `arg3` could be used for `start-point` and `arg4` for `end-point`, but this might vary for each predicate.
- The label `argM` is used for denoting elements that are not arguments selected by the predicate (they are adjuncts or modifiers), and could be further subcategorized marking the type of modifier (i.e. `argM-LOC` or `argM-TMP`).

In our example, the subject is the `agent` and the direct object is the `theme`, so in PropBank notation it would look like the following:

$$[_{arg0}\textit{Juan}]\ \underline{\textit{persiguió}}_{ARG}\ [_{arg1}\textit{un ratón}]$$

Semantic roles are useful for defining a semantic representation that is invariable with respect to the syntactic functions of the constituents. For example, consider the passive voice version of sentence 16:

(17) *Un ratón fue persiguido por Juan / A mouse was chased by Juan*

In sentence 17, the subject and the direct object are switched with respect to sentence 16, but the semantic roles assigned to the participants would be the same:

$$[_{arg1}\textit{Un ratón}]\ \underline{\textit{fue persiguido}}_{ARG}\ [_{arg0}\textit{por Juan}]$$

Semantic Role Labeling

The task of assigning the correct semantic roles to a sentence is denominated **Semantic Role Labeling**. It is a well studied problem in NLP and has been the subject of several competitions, for example [Carreras and Màrquez, 2005] and [Hajič et al., 2009].

The semantic roles representation has the advantage of being very simple. Although it is not as powerful as other semantic representations, it is often used as a base for them, as we will see next.

Abstract Meaning Representation

Abstract Meaning Representation [Banarescu et al., 2013] is a formalism that models a sentence into a graph of participants and actions that relates the participants. It uses the PropBank notation for establishing the relations between predicates and arguments, and it improves over the simple PropBank representation in that the participants can be reused as arguments for different predicates in the graph, and there are more expressive representations for relative and subordinate sentences.

For example, sentence 16 would be modeled as shown in figure 2.8. Notice that all content words in the sentence, the participants and the action, are represented by nodes in the graph, but not the function words. There is an arc from the action to every participant labeled with the semantic role.

```
(p / perseguir-01
  :ARG0 (j / juan)
  :ARG1 (r / ratón))
```

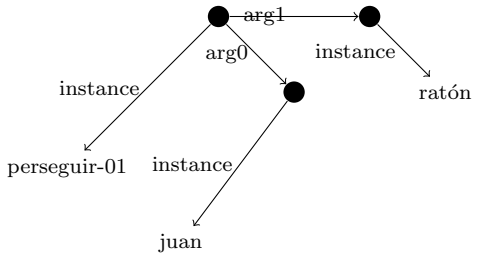


Figure 2.8: AMR text and graph representation for *"Juan persiguió un ratón"*.

Let us consider a slightly more complex sentences that uses a modal verb:

(18) *María quiere comer sushi / Mary wants to eat sushi*

Sentence 18 would be modeled as shown in figure 2.9. In this case, the node that represents *"María"* would be reused as `arg0` of both actions `querer-01` and `comer-01`, but the `arg1` of `querer-01` would be set as the action `comer-01`.

```
(q / querer-01
  :ARG0 (m / maría)
  :ARG1 (c / comer-01
          :ARG0 m
          :ARG1 (s / sushi)))
```

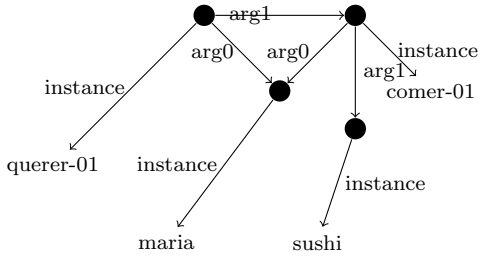


Figure 2.9: AMR text and graph representation for *"María quiere comer sushi"*.

One thing left out in the representation of AMR is the use of universal quantifiers like *"todo"* (*"all"*), which have their own sets of modeling complexities in natural language, as we will see in the following section.

Minimal Recursion Semantics

Minimal Recursion Semantics [Copestake et al., 2005] is a semantics representation formalism that, amongst other things, tries to model the underspecification present in some natural language constructions. Consider the following sample sentence:

(19) *Todos los niños leen un libro / Every kid reads a book*

If we are using first-order logic as a semantic representation, we would see that there are two possible interpretations of this sentence as logic expressions:

$$\exists x \cdot libro(x) \wedge \forall y \cdot niño(y) \rightarrow lee(y, x)$$

Which means, there is a book which all the kids read (they all read the same book). The other possible interpretation is:

$$\forall y \cdot niño(x) \rightarrow \exists x \cdot libro(y) \wedge lee(y, x)$$

Quantifier scope ambiguity

In this case it means each kid reads a book, and all the books are potentially different for each kid. This is an example of **quantifier scope ambiguity**, because it is introduced by the interaction between the universal (*"todos"*) and existential (*"un"*) quantifiers in the sentence. Notice that the both the Spanish and English versions of this sentence present the same ambiguity, and it also happens with other natural languages. This kind of ambiguity is one of the language features that MRS tries to model: in this case the representation will indicate that the precedence of one quantifier over the other is underspecified, so without further information from the text it will remain ambiguous.

The representation of this sentence in MRS would include a set of fixed expressions for $h1 : niño(x)$, $h2 : libro(y)$ and $h3 : lee(y, x)$, and a set of expressions with variable handles for the quantifiers $h4 : todo(x, h1, h6)$ and $h5 : un(y, h2, h7)$. This way, the quantifier $h4 : todo$ is applied to $h1 : niño$, the quantifier $h5 : un$ is applied to $h2 : libro$, but the precedence between $h4$ and $h5$ remains underspecified with handles $h6$ and $h7$. In order to specify one of the two possible meanings, some constraints should be added to these handles, so that either $h6 = h5 \wedge h5 = h3$ or $h6 = h3 \wedge h5 = h4$.

MRS is the semantic representation used in many HPSG frameworks like DELPH-IN, which we will see later on, as it can be implemented as a series of constraints combined using unification during the parsing process.

2.2 Parsing algorithms

As mentioned before, the process of transforming a sentence into its representation in a grammar formalism is called parsing. In this section we will review one of the most important classic parsing algorithms used for CFG called CKY, and present a technique to use this parsing algorithm in the context of deep grammars. See appendix B for a description of other classic parsing algorithms for CFG and dependencies that, like CKY, could be used as a basis for other more complex algorithms we will discuss later.

2.2.1 Cocke-Kasami-Younger

The Cocke-Kasami-Younger (CKY) algorithm [Kasami, 1966; Younger, 1967; Cocke, 1969] is a dynamic programming algorithm that can be used to recognize if a sentence is generated by a grammar in polynomial time. The algorithm works with grammars in the Chomsky Normal Form (CNF). This means all the rules in the grammar must be either binary non-terminal to non-terminal rules ($A \rightarrow B, C$) or unary non-terminal to terminal rules ($A \rightarrow w$) (the pre-terminal rules).

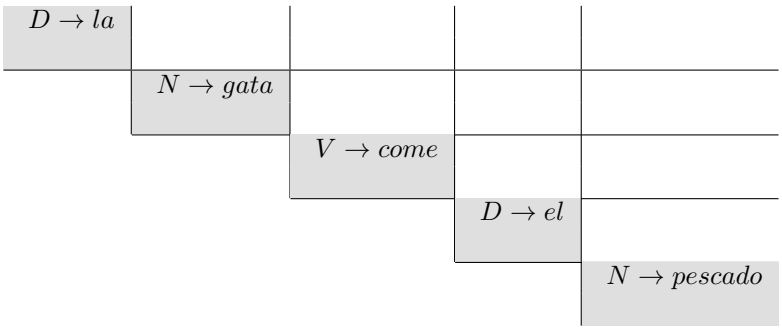
Given this setting, the CKY parser will take a sentence (sequence of tokens) of length n and iteratively fill the upper triangular part of a square matrix M of size $n \cdot n$, in the following way:

- The first step fills the cells in the diagonal of the matrix (cells $M[i, i]$) with the pre-terminal rules for all the tokens.
- At each following step, an upper diagonal line will be filled in the matrix. Each cell $M[i, j]$ will contain the possible ways of combining the tokens t from the sentence so that $i \leq t, t \leq j$, like this:
  - Analyze $M[i, k]$ and $M[k + 1, j]$ for $1 \leq k \leq j - 1$
    $\forall\{A \rightarrow \alpha\} \in M[i, k], \forall\{B \rightarrow \beta\} \in M[k + 1, j]$
    if $\exists\{X \rightarrow A\,B\}$, add $\{X \rightarrow A\,B\}$ to $M[i, j]$
- After the algorithm ends, the sentence will be recognized if a rule with the symbol S (a sentence) is found in the cell $< 1, n >$, i.e. the top-right cell of the matrix.

Let us walk through this algorithm using sample sentence 9 *"La gata come el pescado"* (*"The cat eats the fish"*) and the following grammar:

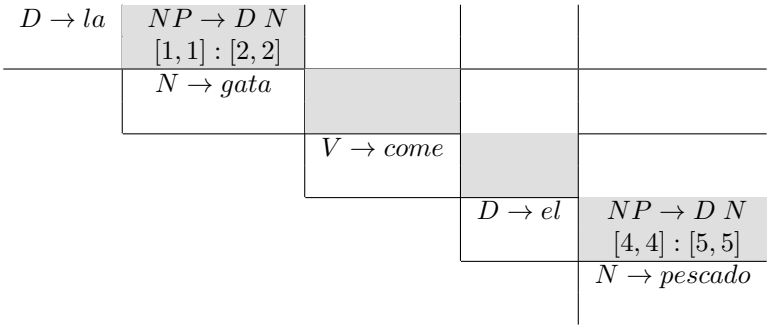
- $R_v = \{D \rightarrow la|el,\ N \rightarrow gata|pescado,\ V \rightarrow come\}$
- $R_g = \{S \rightarrow NP\,VP,\ NP \rightarrow D\,N,\ VP \rightarrow V\,NP\}$

This grammar is very simple and is already in CNF, but it will help us illustrate how the algorithm works. First fill the diagonal with the pre-terminal rules according to the sentence tokens:



In order to fill the next upper diagonal, we start by analyzing cell $M[1,2]$. The objects in this cell should be composed of elements from $M[1,1]$ and $M[2,2]$, which means we should combine D, and N. As there is a rule $NP \rightarrow D\ N$ in the grammar, we add this rule to the cell. We will also leave a mark that indicates which cells participated in this rule application.

The same happens for cell $M[4,5]$, in this case applying $NP \rightarrow D\ N$ to the content of cells $M[4,4]$ and $M[5,5]$. The other cells $M[2,3]$ and $M[3,4]$ do not have elements that can be combined (there are no rules in this grammar to combine $N\ V$ or $V\ D$). The matrix after this step looks as follows:



For the following diagonal, the algorithm will look first at cell $M[1,3]$, and for this cell it will try to combine the elements in the cells $M[1,1]$ with $M[2,3]$ and $M[1,2]$ with $M[3,3]$. As $M[2,3]$ and $M[1,2]$ are empty, no suitable combinations will be found. The same will happen when analyzing cell $M[2,4]$. However, when analyzing cell $M[3,5]$, the algorithm will find that V from $M[3,3]$ can be combined with NP from $M[4,5]$ using rule $VP \rightarrow V\ NP$, thus this rule application is added to $M[3,5]$. The matrix now looks as follows:

$D \to la$	$NP \to D\ N$ $[1,1] : [2,2]$			
	$N \to gata$			
		$V \to come$		$VP \to V\ NP$ $[3,3] : [4,5]$
			$D \to el$	$NP \to D\ N$ $[4,4] : [5,5]$
				$N \to pescado$

The next diagonal will not add any new combinations of elements. First it will look at cell $M[1,4]$, where there are three possible combinations of cells to consider: $M[1,1]$ with $M[2,4]$, $M[1,2]$ with $M[3,4]$ and $M[1,3]$ with $M[4,4]$. There are no suitable elements to combine in those cells, neither in the cells considered for $M[2,5]$, so the matrix will remain the same as in the previous step:

$D \to la$	$NP \to D\ N$ $[1,1] : [2,2]$			
	$N \to gata$			
		$V \to come$		$VP \to V\ NP$ $[3,3] : [4,5]$
			$D \to el$	$NP \to D\ N$ $[4,4] : [5,5]$
				$N \to pescado$

Finally, the last step will only look at one cell $M[1,5]$ and try to fill it with combinations of four different pairs of cells: $M[1,1]$ with $M[2,5]$, $M[1,2]$ with $M[3,5]$, $M[1,3]$ with $M[4,5]$ and $M[1,4]$ with $M[5,5]$. The only pair with suitable elements to combine is $M[1,2]$ (NP) and $M[3,5]$ (VP), so the rule $S \to NP\ VP$ is added to this cell.

$D \to la$	$NP \to D\ N$ $[1,1] : [2,2]$			$S \to NP\ VP$ $[1,2] : [3,5]$
	$N \to gata$			
		$V \to come$		$VP \to V\ NP$ $[3,3] : [4,5]$
			$D \to el$	$NP \to D\ N$ $[4,4] : [5,5]$
				$N \to pescado$

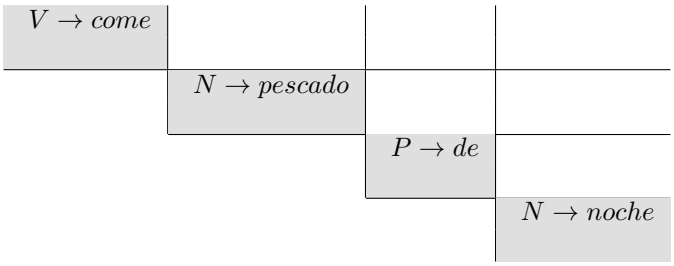
After the algorithm finished filling up the matrix, there is a rule application with left side S in the cell $M[1,5]$, which means the entire sequence could be

recognized as a sentence. Furthermore, if we keep pointers to the different rule applications that were used in each step, we can build the corresponding parse tree for the sentence.

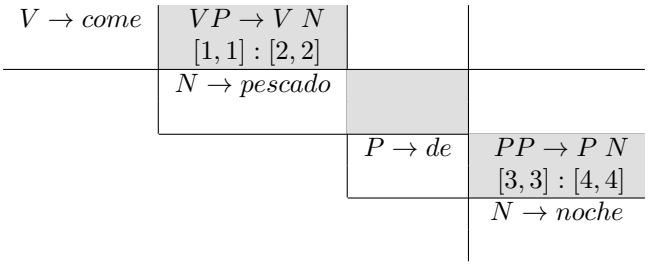
CKY is also good for handling possible ambiguities in the input. Suppose we are trying to parse sample sentence 12 *"La gata come pescado de noche"*, let us focus on the application of the algorithm to the subsequence *"come pescado de noche"* (*"eats fish in the night"*) using the following grammar extract:

- $R_v = \{\ldots N \rightarrow pescado|noche,\ V \rightarrow come,\ P \rightarrow de\}$
- $R_g = \{\ldots NP \rightarrow N\ PP,\ PP \rightarrow P\ N$
  $VP \rightarrow V\ N,\ VP \rightarrow V\ NP,\ VP \rightarrow VP\ PP\}$

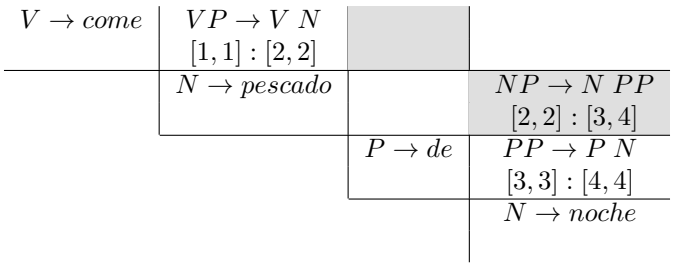
We first fill the diagonal in the matrix with the pre-terminals:



In the following step, cells $M[1,1]$ and $M[2,2]$ are combined using the $VP \rightarrow V\ N$ rule; and cells $M[3,3]$ and $M[4,4]$ are combined using the $PP \rightarrow P\ N$ rule.



The next step will combine cells $M[2,2]$ *"pescado"* and $M[3,4]$ *"de noche"* to form a possible noun phrase.



When getting to the final step, there will now be two ways of generating the VP: first it is possible to combine $M[1,2]$ *"come pescado"* with $M[3,4]$ *"de noche"*; the other option is combining $M[1,1]$ *"come"* with the NP created in the previous step $M[2,4]$ *"pescado de noche"*. For a human, the preferred option would certainly be the first one, but the ambiguity of this sequence with respect to this grammar is real, and CKY is able to capture all possible analysis and pack them into its matrix, as shown below:

$V \rightarrow come$	$VP \rightarrow V\ N$ $[1,1] : [2,2]$		$VP \rightarrow VP\ PP$ $[1,2] : [3,4]$ $VP \rightarrow V\ NP$ $[1,1] : [2,4]$
	$N \rightarrow pescado$		$NP \rightarrow N\ PP$ $[2,2] : [3,4]$
		$P \rightarrow de$	$PP \rightarrow P\ N$ $[3,3] : [4,4]$
			$N \rightarrow noche$

When the length of the sentence and the complexity of the grammar grow, there will probably be many possible interpretations for a single sentence, and not all of them would be equally likely. One way of dealing with this would be using a PCFG (see section 2.1.2). It is possible to include the calculation of probabilities at each step during the CKY algorithm to in the end we can get each tree associated with its probability. As well as this, when the number of possible parse trees becomes too large, it is also possible to use a pruning strategy based on the partial probabilities found for the subtrees in each cell. This strategy is not perfect, as in some cases it is possible that the most likely tree might not be found, but it is generally a good strategy for speeding up the execution time of CKY when the sequences are too ambiguous and still getting trees with high probability.

CKY can also be used with slight adaptations for parsing using grammar formalisms different than CFG, such as CCG or HPSG, if we make sure that the possible rules to apply are always binary.

2.2.2 Supertagging

Deep formalisms like CCG, TAG or HPSG are all highly lexicalized, which means much of the combinatorial information that would be used during the parsing process is encoded in the categories associated to the words. This has the consequence that in general there will be multiple possible categories that a word could have, because the same word in different contexts might behave differently. Trying with hundreds of different categories for each word in a sentence renders the parsing problem intractable, so having the correct category for each word is essential for these deep grammars.

Because of this, one very common approach to parsing with deep grammars involves the following steps:

- Find the most likely categories for each word.
- Use a parsing algorithm like CKY or chart parsing to find the best tree given those categories.

Supertagging

The first step is called **supertagging**. It is an extension of the task of part of speech tagging, but instead of returning a simple category like a part of speech, it returns a fine grained label that contains more information. Using these fine grained labels, called **supertags**, one might reconstruct the appropriate lexical entry for each word that will be used in the rest of the analysis. For example, consider we want to parse sample sentence 2 into HPSG format:

Supertag

la gata negra duerme

First we need to use a supertagger to guess the appropriate lexical entries for each word. The supertagger would return a supertag for each word in the sentence, like the following:

la/`d-x` *gata*/`n-sd-x` *negra*/`a-mn-x` *duerme*/`v-sn-x`

The supertag format used in this example is the one we are going to use in this work. It will be defined in detail in section 5.4, but for now we can say that each tag indicates the part of speech followed by a description of the HPSG features that the lexical entry has. For example, the tag for *"negra"*, `a-mn-x`, starts with "`a`" meaning that it is an adjective, followed by "`mn`" which means that the lexical entry should have a `MOD` feature expecting something of type `n` (like a noun phrase). From these supertags, it is easy to build the corresponding lexical entries like the ones shown in figure 2.5. Then it is possible to use an algorithm like CKY to find the correct tree.

Creating a supertagger often involves using machine learning to train a system from a corpus of examples. The task of supertagging was initially proposed in the context of lexicalized TAG parsing [Joshi and Srinivas, 1994; Kasai et al., 2017; Friedman et al., 2017] but has later on been applied for other deep formalisms like CCG [Curran et al., 2006; Lewis and Steedman, 2014] or HPSG [Matsuzaki et al., 2007; Dridan, 2009; Zhang et al., 2010]. The main aim of the process is to limit the combinatorial explosion that happens in this kind of grammars due to the high ambiguity of the lexical entries.

Supertagging is not a parsing algorithm in itself, but depending on the level of detail contained in the supertags, the rest of the parsing process might be simplified or even made trivial. Because of this the process has sometimes been referred to as "almost parsing" [Joshi and Srinivas, 1994].

2.3 HPSG implementations

Originally proposed for English [Pollard and Sag, 1994] but with possible extensions to other languages, HPSG has had some variants and differences in implementations throughout the years. The main characteristics of the grammar shared by all implementations include the use of feature structures for modeling words, phrases and whole sentences, but the concrete set of features used varies from implementation to implementation.

For example, in the original grammar from Pollard and Sag [Pollard and Sag, 1994], there were different valence features for indicating the subject of a verb and the determiner attached to a noun in a noun phrase. This was changed in future revisions [Sag et al., 2003] so as to use an approach more similar to X' [Chomsky, 1995], in which both the subject of a sentence and the determiner of a noun phrase are considered the specifiers of their corresponding phrases. Some of the HPSG implementations are more closely based on one model or the other, and tend to use the features defined in that model. These differences are accentuated when creating different grammars for languages other than English, because the languages vary on agreement, word order, use of affixes, and many other characteristics. There are also differences in the way the implementations represent semantics.

We will explain some concepts of the two more widely used implementations of HPSG grammars: the DELPH-IN framework and the Enju family of parsers. We will discuss characteristics of these two implementations for English, and one example for Spanish, although this does not mean to be an exhaustive representation of all HPSG implementations. In our case, our own implementation of HPSG will share elements with both (see chapter 3), but also some adaptations to the language and some simplifications derived from the resources we used to build it.

2.3.1 DELPH-IN

DELPH-IN[3] (short for Deep Linguistic Processing with HPSG Iniciative) is an initiative to create HPSG grammars for different languages, so that the created grammars share certain characteristics that allow some level of interoperability. This interoperability could be used, for example, to support syntactic and semantic based machine translation between the participant languages [Bond et al., 2005].

One of the projects associated to DELPH-IN is the LinGO Grammar Matrix [Bender et al., 2002], a framework for creating HPSG grammars that has been used for creating Norwegian, Korean, Portuguese and Spanish versions of the grammar, amongst other languages. DELPH-IN also includes the Linguistic Knowledge Builder (LKB) system [Copestake et al., 1999; Copestake, 2002], an environment for implementing constraint-based grammars that is widely used in the development of HPSG grammars. The parsing process used by this system

[3] `http://www.delph-in.net/wiki/index.php/Home`

is based on a chart parser (see appendix B.1). Besides LKB, DELPH-IN also includes another system for working with unification grammars denominated PET [Callmeier, 2000], which is based on the same language as LKB and includes a re-ranking module for classifying the possible parse results according to a statistical model selection.

There exists another more modern parser that can be used with the same language for creating grammars in LKB which is the Answer Constraint Engine (ACE parser) [Packard, 2013]. It includes built-in support for part of speech tagging and unknown words handling, and it is reported to run up to fifteen times faster than the LKB parser (on par and sometimes exceeding the PET parser performance).

The *de facto* standard implementations of English and Spanish HPSG grammars are built in the DELPH-IN framework. The reference implementation for English HPSG is the English Resource Grammar (ERG)[4] [Copestake and Flickinger, 2000], a grammar that was manually designed with the aim of providing very detailed and rich analysis for complex sentences. All feature structures, grammar rules, type hierarchy and lexical entries were hand crafted. An example of a valid sentence that can be analyzed using this grammar is:

So if we can not make it on Thursday afternoon, we will have to, you know, look for something.

With this example, the tool would yield 92 possible analyses for the sentence. The grammar also tries to be very precise in letting out ungrammatical sentences like the following:

* *How does Wednesday the twenty fourth after one o'clock?*

If was first applied in the machine translation project Verbmobil for translating between English, German and Japanese. The use of MRS for semantic representation was one of the strengths to use in this translation project oriented to semantic transfer. For that first project the first 5,000 lexical entries were hand written, but later on the project has grown in size and coverage of the language (see section 2.4.4).

The ERG follows closely the HPSG description found in [Sag et al., 2003]. It also uses the proposed semantic representation, minimal recursion semantics, with the following characteristics:

- Each word marked as predicate defines a set of semantic arguments as features.
- Compositional semantics: the parsing process creates a list of semantic restrictions by concatenating the restrictions of each predicate and their associated arguments.
- There is a feature that describes the semantic mode of a predicate: proposition, question, directive, reference.
- It takes into account semantics of quantifiers and their scopes.

[4] `http://www.delph-in.net/erg/`

2.3.2 Spanish Resource Grammar

The Spanish implementation of HPSG contained in the LinGO framework is denominated the Spanish Resource Grammar (SRG) [Marimon, 2010]. It is to date the most complete HPSG grammar developed for Spanish, containing more than 50,000 lexical entries, 64 lexical rules and 191 combinatorial rules for forming phrases. The rules were developed manually and they used manual and semi-automatic processes to build the lexicon [Marimon et al., 2007]. As part of the DELPH-IN initiative and the LinGO matrix, SRG also uses the MRS semantics representation. The parse trees obtained using SRG are very rich trees that support many of the linguistic constructions provided by the HPSG theory.

Due to the use of the LKB parser, the original version of this grammar did not include statistical analysis support, for example for tree disambiguation. However, later on a disambiguation post-processing was added [Marimon et al., 2014b], created with an iterative process together with the construction of the IULA corpus (see section 2.4.5).

In this work we focus on a different approach for building a Spanish HPSG grammar: beginning with an annotated corpus, and using this information to extract the lexical entries and perform a statistical modeling of the grammar rules and lexical entries. This approach might be more flexible, but we must take into account that it will rely strongly on the content of the original corpus and how well it represents the language.

2.3.3 Enju

The grammar development strategy we follow in this work is inspired by the Enju system[5] [Ninomiya et al., 2006], a statistical HPSG parser with high coverage for the English language. Enju is a high performance parser that also successfully resolves some complex linguistic phenomena like raising verbs and control verbs, and also has some success at analyzing coordinated phrases. It also has versions for Japanese and Chinese.

For building this parser, they followed a radically different approach than the one used for the ERG. Instead of manually crafting the grammar rules and lexical entries, they created an HPSG corpus based on the Penn Treebank corpus (see section 2.4.1). As the syntactic annotations of this corpus are not directly comaptible with an HPSG grammar, they implemented a conversion process that adapts the Penn Treebank rules to a format similar to HPSG [Miyao et al., 2005]. Then they used the resulting corpus to build the lexical entries and extract the rule application probabilities.

The Enju framework includes two parsers:

- The eponymous Enju parser first uses a supertagger to obtain the most likely lexical entries for the words in a sentence and applies a statistical CKY algorithm (see section 2.2.1) with these lexical entries to obtain the

[5] `https://mynlp.is.s.u-tokyo.ac.jp/enju/index.html`

most likely parse tree [Miyao and Tsujii, 2005]. This process returns the parse trees with high accuracy, although it could be a little slower than the other parser (around 500 ms per sentence).

- The Mogura parser uses the same supertagger as Enju, but it only keeps the most likely entry for each word and uses a shift-reduce parser (see appendix B.2) for building the final tree [Matsuzaki et al., 2007]. This process is much faster (around 50 ms per sentence) but in some cases the analyses might not be the best possible.

The supertagger built for Enju [Zhang et al., 2010], and used by both parsers Enju and Mogura, uses a CFG that approximates the Enju HPSG grammar. This CFG allows to quickly discard invalid analyses, information that is incorporated during the supertagger training in order to improve its precision. This supertagger achieves an accuracy of 93.98%.

Enju does not use the LKB platform for working with feature structures like the grammars supported by DELPH-IN do. Instead, they developed their own language called LiLFeS [Takaki et al., 1998]. LiLFeS is a programming language based on the logic programming paradigm where the fundamental data type is the typed feature structure (TFS).

Unlike the English Resource Grammar, the grammar used by the Enju parser incorporates some more elements of the HPSG version in [Pollard and Sag, 1994]. Their approach for semantics is also different than DELPH-IN as the representation they use is more akin to standard semantic role labeling. Instead of modeling semantics using MRS, they just include features for modeling the semantic roles in the PropBank notation [Bonial et al., 2010]. This is a simpler approach that lacks the compositional restrictions and more complex modeling found in MRS.

Enju's approach of transforming a constituency treebank into HPSG notation and use this conversion for training a parser was also followed more recently in the work of [Zhou and Zhao, 2019], which combined the constituency and dependency versions of the Penn Treebank to infer HPSG annotations. They were also inspired by the original Enju grammar but decided to use their own transformation of the corpus. The parser they built obtains very good performance over the Penn Treebank (see section 2.7).

2.4 Treebanks

Treebanks are collections of sentences annotated with syntactic information in some formalism. They are an essential component for parser development and for statistical analysis. Treebanks differ in their language, size, and the formalism they are annotated in.

2.4.1 Penn Treebank

The Penn Treebank [Marcus et al., 1993] is a corpus of English language sentences of about 4.5 million words. It contains texts from different sources, each section annotated with different types of annotations including parts of speech, parse trees, predicate-argument structure, or speech disfluencies [Taylor et al., 2003]. This corpus is widely used for statistical parser development because it contains a large part annotated with syntactical annotations consistent with a CFG-style grammar. This section consists in articles from the Wall Street Journal (WSJ) and comprises 25 sections, where sections 0-18 (about 910K words) are generally used for training, 19-21 (around 130K words) for development and 22-24 (around 130K words) for testing purposes.

As the syntactic trees present in the Penn Treebank include generally rather flat structures (called skeletal parses), the CFG induced by these trees contains a great number of rules. There are many rules for each non-terminal, and some of them might be very long. For example, there are 4,500 rules for VPs, and the whole WSJ section contains 17,500 CFG-style rules [Jurafsky and Martin, 2014].

Figure 2.10 contains an extract of one sentence from the Penn Treebank annotated in its syntactic format. The full sentence without markup reads: *"Mr. Wathen, who started his career as an Air Force investigator and worked as a security officer for several large companies, built his California Plant Protection from a tiny mom-and-pop security patrol firm here in the San Fernando Valley."* Notice that even for this sentence there are already twelve different ways of writing a NP:

- $NP \rightarrow NNP\ NNP$
- $NP \rightarrow$ -$NONE$-
- $NP \rightarrow PRP\$\ NN$
- $NP \rightarrow DT\ NNP\ NNP\ NN$
- $NP \rightarrow NP\ PP$
- $NP \rightarrow DT\ NN\ NN$
- $NP \rightarrow JJ\ JJ\ NNS$
- $NP \rightarrow NP\ ,\ SBAR\ ,$

```
( (S
    (NP-SBJ
      (NP (NNP Mr.) (NNP Wathen) )
      (, ,)
      (SBAR
        (WHNP-14 (WP who) )
        (S
          (NP-SBJ (-NONE- *T*-14) )
          (VP
            (VP (VBD started)
              (NP
                (NP (PRP$ his) (NN career) )
                (PP (IN as)
                  (NP (DT an) (NNP Air) (NNP Force) (NN investigator) ))))
            (CC and)
            (VP (VBD worked)
              (PP-CLR (IN as)
                (NP (DT a) (NN security) (NN officer) ))
              (PP-CLR (IN for)
                (NP (JJ several) (JJ large) (NNS companies) ))))))
      (, ,) )
    (VP (VBD built)
      (NP (PRP$ his) (NNP California) (NNP Plant) (NNP Protection) )
      (PP-CLR (IN from)
        (NP
          (NP (DT a) (JJ tiny) (JJ mom-and-pop) (NN security) (NN patrol) (NN firm) )
          (ADVP-LOC (RB here)
            (PP (IN in)
              (NP (DT the) (NNP San) (NNP Fernando) (NNP Valley) ))))))
    (. .) ))
```

Figure 2.10: Extract from the WSJ section of the Penn Treebank.

- $NP \rightarrow PRP\$\ NNP\ NNP\ NNP$
- $NP \rightarrow DT\ JJ\ JJ\ NN\ NN\ NN$
- $NP \rightarrow DT\ NNP\ NNP\ NNP$
- $NP \rightarrow NP\ ADVP$

Penn Treebank includes some marks after the grammar categories for indicating attributes about the syntactic structure. For example, some of the NPs are marked as -SBJ, indicating they are the subjects inside their respective clauses.

It also includes some discontinuity markers, for example the subject of the relative sentence *"started his career as an Air Force investigator and worked as a security officer for several large companies"* is marked as -14, which is coindexed with the previous $WHNP$ *"who"*.

This information is very useful for the development of deep grammars. For example [Miyao et al., 2005] and [Zhou and Zhao, 2019] use a rule-based transformation process for converting this corpus into an HPSG compatible format,

and it has also been used for CCG grammar development [Hockenmaier and Steedman, 2007].

2.4.2 AnCora

AnCora [Taulé et al., 2008] is a corpus of Spanish and Catalan texts of approximately 500,000 words. It contains 17,000 sentences in around 1,600 news articles. All sentences are annotated syntactically in a CFG-style format. Words and constituents also incorporate more information as XML attributes, for example:

- Morphological attributes: The parts of speech and morphological information of the words are encoded using the EAGLES tagset[6]. Besides this, AnCora also encodes each morphological feature as a separate attribute in every word.
- Grammatical categories of constituents: There is an extensive set of categories including `grup.verb` for verbal periphrases, `S` for subordinate sentences, `sn` for noun phrases and `grup.nom` for noun groups (noun phrases that lack a specifier). There are many singleton categories such as `infinitiu` for infinitive verbs, `participi` for participles and `prep` for prepositions that are generally redundant as they do not add extra attributes. Also, the annotation is not completely consistent across the whole corpus.
- Semantic argument structure: They use the `arg` attribute for marking semantic roles using the PropBank notation. Values `arg0` through `arg4` are semantic arguments, `argM` are adjunct and `argL` are arguments of light verbs (e.g. *"las gracias"* in *"dio las gracias"* / *"thanked"*).
- WordNet senses: This is encoded as the attribute `sense`, but is annotated for nouns only.
- Verbal subcategorization: Encoded in the lexical semantic structure (`lss`) attribute. There are thousands of different subcategorization frames for verbs classified in four different categories (accomplishments, achievements, states and activities) [Aparicio et al., 2008].
- Null subjects: Instances of null subjects in the corpus, which are very common in Spanish, are marked as a noun phrase `sn` with an attribute `elliptic='yes'`. This is useful as this structure is marked as subject and has the corresponding semantic role when available.

The process for building AnCora used an automatic morphosyntactic and surface chunking method and based on this information the annotators performed a manual labeling of the deep structure of the sentences [Martí et al., 2007].

[6] `http://blade10.cs.upc.edu/freeling-old/doc/tagsets/tagset-es.html`

Besides the annotated corpus, there are also two lexical resources associated to AnCora: AnCora-Verb [Aparicio et al., 2008] contains 2,647 verbs for Spanish and their corresponding subcategorization frames. AnCora-Nom [Peris Morant and Taulé Delor, 2011] contains 1,600 Spanish deverbal nouns and their corresponding subcategorization frames.

```
<sentence>
  <sn tem='tem' func='suj' arg='arg1'>
    <spec> <d wd='La' pos='da0fs0' lem='el'/> </spec>
    <grup.nom>
      <s.a func='cn'>
        <grup.a> <a wd='próxima' pos='aq0fs0' lem='próximo'/> </grup.a>
      </s.a>
      <n wd='apuesta' lem='apuesta' pos='ncfs000'/>
      <sp tem='agt' func='cn' arg='arg0'>
        <prep> <s wd='del' pos='spcms' lem='del'/> </prep>
        <sn> <grup.nom> <n wd='Gobierno' pos='np00000' lem='Gobierno'/> </grup.nom> </sn>
      </sp>
      <S func='cn'>
        <f wd=',' lem=',' pos='fc'/>
        <infinitiu> <v wd='reformar' pos='vmn0000' lem='reformar'/> </infinitiu>
        <sn tem='tem' func='cd' arg='arg1'>
          <spec> <d wd='la' pos='da0fs0' lem='el'/> </spec>
          <grup.nom>
            <n wd='negociación' pos='ncfs000' lem='negociación'/>
            <s.a func='cn'>
              <grup.a> <a wd='colectiva' pos='aq0fs0' lem='colectivo'/> </grup.a>
            </s.a>
          </grup.nom>
        </sn>
        <f wd=',' lem=',' pos='fc'/>
      </S>
    </grup.nom>
  </sn>
  <grup.verb>
    <v wd='puede' pos='vmip3s0' lem='poder'/>
    <infinitiu> <v wd='ser' pos='vsn0000' lem='ser'/> </infinitiu>
  </grup.verb>
  <sn tem='atr' func='atr' arg='arg2'>
    <spec> <d wd='el' pos='da0ms0' lem='el'/> </spec>
    <grup.nom>
      <n wd='reactivo' pos='ncms000' lem='reactivo'/>
      <sp>
        <prep> <s wd='para' lem='para' pos='sps00'/> </prep>
        <S clausetype='completive'>
          <infinitiu> <v wd='recuperar' pos='vmn0000' lem='recuperar'/> </infinitiu>
          <sn tem='pat' func='cd' arg='arg1'>
            <spec> <d wd='la' pos='da0fs0' lem='el'/> </spec>
            <grup.nom>
              <s.a> <grup.a> <a wd='deseable' pos='aq0cs0' lem='deseable'/> </grup.a> </s.a>
              <n wd='unidad' pos='ncfs000' lem='unidad'/>
              <s.a>
                <grup.a> <a wd='sindical' pos='aq0cs0' lem='sindical'/> </grup.a>
              </s.a>
            </grup.nom>
          </sn>
        </S>
      </sp>
    </grup.nom>
  </sn>
  <f wd='.' lem='.' pos='fp'/>
</sentence>
```

Figure 2.11: Simplified extract from the Spanish AnCora corpus.

Figure 2.11 shows a simplified example of a sentence from the Spanish AnCora corpus. The words of the sentence are encoded in the attribute `wd` of the elements whose tags correspond to one of the POS-tags (`n`, `v`, `a`, etc.). The full sentence reads: *"La próxima apuesta del Gobierno, reformar la negociación colectiva, puede ser el reactivo para recuperar la deseable unidad sindical."* We simplified the example by removing most of the attributes (redundant morphological information, coreferences, subcategorization types, noun senses, etc.), but we kept the full structure of categories.

Compared to Penn Treebank, the structure of AnCora is more complex in terms of categories and in the level of elements nesting. For example, a noun phrase (`sn`) usually will contain a nominal group (`grup.nom`) and optionally a specifier (`spec`). The `grup.nom` will contain the main noun (`n`) of the phrase and other modifiers such as adjectival phrases (`s.a` or `sa`) or prepositional phrases (`sp`).

This structure is honored in many cases, but not always, as there are some inconsistencies in the annotation in some parts of the corpus. Because of this, it happens that, like the Penn Treebank, there is a great number of rules per category and a huge diversity in the way the rules are used. For example, in the corpus there are 5,826 rules for subordinate sentences (`S`), 2,403 for sentences (`sentence`), 905 for nominal groups (`grup.nom`) and 295 for noun phrases (`sn`).

2.4.3 Universal Dependencies

Universal Dependencies[7] [Nivre et al., 2016] is a framework for annotating dependency grammars across many languages in a unified format. They also include a standard unified notation for parts of speech and morphological attributes across all languages.

The latest version of Universal Dependencies includes corpora for 150 languages, although the sizes of these corpora varies greatly between languages. For English and for Spanish there are several corpora, summing up approximately 650 thousand tokens in 37,000 sentences for English, and one million tokens in 35,000 sentences for Spanish. One of the Spanish corpora is the conversion of AnCora to the Universal Depedencies format.

One of the aims of the project is trying to have a unified format that would apply to all languages. This could be used to improve the quality of the information used in multilingual parsing, an idea that has been explored in some parsing challenges [Zeman et al., 2017, 2018].

This unified representation involves defining the notion of headness in a way rather different than other dependency specifications. For example, some conversions of the Penn Treebank to dependency format define the head of a prepositional phrase to be the preposition. The same happens for some conversions of Spanish AnCora to dependencies. However, Universal Dependencies prefers using content words over function words as heads. One reason for this is that it is easier to create more similar dependency representations for languages that use post-positions instead of prepositions. The way to do this in UD is considering that the main content word is the head and the affix will be marked as a `case`.

UD also has its own strict definition of what is considered a token, and multiword expressions are not considered single tokens as in AnCora. The AnCora conversion to this format [Alonso and Zeman, 2016] involved separating these multi-word expressions and providing adequate analyses for them, for example for verbs with enclitics and proper names. The UD formalism does not provide argument structure information in the form of semantic roles, so this information is lost from the original AnCora. Furthermore, given that the original AnCora tokens were transformed, the map between the semantic roles information in AnCora and its UD version is not straightforward.

[7] `https://universaldependencies.org/`

2.4.4 DeepBank

DeepBank is a version in HPSG format of all 25 WSJ sections of Penn Treebank (around 50,000 sentences) [Flickinger et al., 2012][8]. They parsed the sentences using the English Resource Grammar over LKB and PET.

The corpus was developed through an iterative process. They used PET to find the 500 most likely analyses according to the grammar, manually picking the best one, then updating the parser with the new information. The grammar might occasionally be updated to improve the coverage of the rules based on new information, and the existing trees needed to be re-parsed after that.

Not all the sentences have a representation in the corpus. Around 15% of the sentences could not be parsed completely using the HPSG grammar mainly because of these reasons:

- Some of the sentences contained errors in the original corpus.
- Even after the corrections performed to the ERG, the grammar might still have some representation gaps, and some rare constructions might not be fully analyzed.
- There are limits in the computational resources. Sometimes the correct tree is not amongst the most likely trees found by the PET parser.

The final corpus contains full HPSG derivations together with their MRS semantic representation for approximately 85% of the sentences. They completed the representation of the rest of the corpus using a PCFG that approximates the behavior of ERG.

2.4.5 IULA

The IULA Treebank[9] [Marimon et al., 2012, 2014a] is a corpus of Spanish sentences annotated in a dependency grammar style, containing about 590,000 words in 42,000 sentences. The development of this corpus was done together with the development of the statistical disambiguation module for SRG [Marimon et al., 2014b]. The process for developing this corpus was similar to the one used for DeepBank, with some differences:

- Get all the possible analyses for the sentences using the LKB parser with the SRG grammar.
- Manually pick the best analysis for each sentence.
- Train two models over the already parsed sentences (a maximum entropy model and one based on dependency-like features) for predicting the best parse tree.

[8] `http://moin.delph-in.net/DeepBank`
[9] `http://www.iula.upf.edu/recurs01_tbk_uk.htm`

- If both models agree on the correct parse tree for a sentence, pick that tree, or else mark the sentence to be manually disambiguated by an annotator.

Using this process, the authors could build a large corpus starting with a relatively small set of manually disambiguated parse trees, building a larger corpus together with more accurate classifiers with each iteration. The final corpus was transformed into dependency format for publication. However, the original HPSG version of the corpus is not available for download.

Compared to AnCora, the IULA Treebank generally contains shorter sentences: the longest sentence in IULA has 33 words, while AnCora has sentences longer than a hundred words. It contains a lot of technical language, as it was built using sentences from documents about law, economy, genomics, medicine, and environment. One problem with this corpus is that, unlike AnCora, it does not provide readily available argument structure information.

There was another attempt at creating a Spanish HSPG corpus based on AnCora using the same idea as DeepBank: the Tibidabo Treebank [Marimon Felipe, 2010]. However, this treebank also has limitations, as only sentences less that 40 words long could be converted, and not all of them could be parsed into the HPSG format.

2.5 Machine Learning

The term machine learning [Mitchell, 1997] refers to the study of algorithms that improve their performance automatically through experience. These algorithms are usually statistical techniques that take a set of data as input and generate a model, this model could be then used to guide the behavior of the algorithm on new unseen data. The main machine learning approaches can be categorized as:

- Supervised learning: The system has a set of inputs associated with the corresponding expected outputs, so the algorithm will try to create a model that predicts outputs on unseen data.
- Unsupervised learning: The system has a set of inputs but no expected outputs, so the algorithm tries to discover correlations in data without having an explicit goal on what to find.
- Reinforcement learning: The system can make decisions at some points during the execution, and some sequences of decisions lead to positive or negative feedback, the algorithm tries to learn how to maximize the positive outcomes while minimizing the negative ones.

In this work we will deal mainly with supervised techniques. Supervised machine learning problems can be further subdivided in regression or classification problems. Regression problems are those in which the output values belong to a continuum, for example obtaining the price of a house given a series of features like size, number of bedrooms, neighborhood, etc.

Classification problems are those in which the output is a value from a discrete set of classes. For example, given a tweet, classify whether the tweet has positive, negative or neutral sentiment.

For some problems, the possible outputs are not simple classes but convey structured information, like a list or a tree of values, which are in some way linked to the input object. Parsing is one of these problems with structured outputs. The output could be, for example, a tree where each node is a category, and in which the leaves are the words of the input sentence.

Annotated corpus

Training

As mentioned earlier, supervised learning always relies on having a set of input data for which we already know the corresponding outputs. In NLP, the input data generally is an **annotated (or labeled) corpus** of samples. When we use a machine learning method to create a statistical model of the data we say we are **training** the model. For supervised classification problems, we usually call these models classifiers.

2.5.1 Methodology

When using machine learning, our aim is to find a model, based on the available data, that can predict the outcome over unseen data. There are many supervised machine learning methods, like Naïve Bayes, Support Vector Machines, Logistic

Regression or the nowadays ubiquitous Artificial Neural Networks, that could be used for this. These methods differ in the kind of problems they can tackle, the amount of data and computing power needed to train them, their generalization capacity, and other factors. But besides selecting the method, each one has its own set of **hyperparameters**: parameters (different than the input data) that modify the training process and which should be tuned to get optimal performance. Given this wide range of possibilities, we need a methodology to guide us into finding, if not the best possible model, at least a model good enough to suit our needs.

Hyperparameters

As we want the model to be able to predict the outcome on unseen data, we need to keep a set of data that the method has not used for training. The usual way to do this is to split the original annotated corpus we have in two partitions: one of them is going to be used to train the model (**training corpus**) while the other is going to be used to evaluate how well the model behaves on data that it has not been trained on (**test corpus**). If we used the same set of data for training and testing, then the method could just be memorizing the training data samples and not generalizing. We want to find a model that generalizes as much as possible to unseen data, and keeping the training and test data separated is essential to check if the model has generalized.

Training corpus

Test corpus

When a model that has been trained on some data has a very low performance when evaluated against that same data, we say the model has **underfit**, and it generally means that the capacity of the model is not enough to learn the differences in the input data. On the other hand, if the model has very high performance on the training data, but has low performance on the test data, we say the model has **overfit**: it learned to make predictions based on particularities of the training data but was not able to generalize to unseen data. As we want a model that has the best possible performance, we want to avoid both of these scenarios. Sometimes the only way to find the best system that does not overfit or underfit implies searching through several machine learning methods and trying many different configurations. Because of this, it is not enough to have only one test set for evaluating the outputs. As we have many possible supervised methods, each one of them with many possible hyperparameters, we are probably going to create a large number of statistical models and keep the one with the best performance. If we test all these models against our only test corpus, there is no way of telling if the results of our best model were achieved because this model is the one that best generalizes on unseen data, or if we just stumbled by chance upon a model that has good results over the test data but does not generalize to other cases.

Underfitting

Overfitting

The way to solve this is not using the test set until the end of the process, and only use it to evaluate the final results of the model (or models) selected. This means that we must evaluate the intermediate models we are generating on other unseen data. There are two techniques that are used for this. One of them is creating a subdivision of the original corpus in three partitions instead of two, adding a new partition that is called the **development corpus** (or sometimes validation corpus), that will be used specifically to evaluate and compare different models and tune hyperparameters. Splitting the corpus in

Development corpus

training, development and test sets is the best solution if there is enough data. However, if the training corpus is already too small, we might use another technique called **cross validation** that entails subdividing the training corpus in n parts. For each model that we want to evaluate, we train n versions of the model using $n-1$ partitions as training data and one partition for validation. Then we average the results of the n models and pick the one that is best on average.

Cross validation

Depending on the machine learning method we use, there might also be other kinds of partitions used to tune the inner parameters of the method. For example it is usual when training a neural network that we further subdivide the training corpus to create a **held-out corpus** (which is also sometimes referred to as validation corpus).

Held-out corpus

2.5.2 Metrics

We mentioned the need to evaluate the models over unseen data, but we have not addressed what it means to evaluate a model yet. Evaluation is also a crucial step in machine learning, as we want the models to improve their performance based on experience, so we need a way to measure this performance.

The evaluation metrics to use will depend on the problem we are trying to solve. Regression and classification problems use different metrics. In this section we will discuss some of the metrics associated to classification, that will serve as basis for the more concrete performance metrics we will discuss in later sections.

Suppose we have a classification problem with binary output. For example given a tweet, the system must tell if the tweet has a positive (class `P`) sentiment or not (class `N`). We train a classifier with the training data using some machine learning method, and then we run the classifier over the test data and obtain a set of outputs. We already know the expected output values for each sample from the test set (which we will call the **gold standard**), and the classifier (whose outputs we will call the **candidates**) could have correctly guessed some of those values, and probably missed some others.

Gold standard

Candidate

		Classifier output	
		P	N
Expected values	P	10	4
	N	2	8

Table 2.2: Example of expected outputs (gold standard) vs. classifier outputs (candidates) for a fictitious classification problem.

Table 2.2 shows an example of counts of expected values against the values output by the classifier. This is called a **confusion matrix**. We will denominate *true positives* the samples that were expected to be positive and the model classified them as so, *true negatives* the samples that were expected to be negative and were correctly classified, *false positives* the samples that were expected

Confusion matrix

to be negative but the model classified as positive, and conversely *false negatives* are the samples that were expected to be positive and the model classified erroneously as negative. We can then define the following metrics:

Precision

$$Precision = \frac{true\_positives}{true\_positives + false\_positives} = \frac{|gold \cap candidates|}{|candidates|} \tag{2.1}$$

The precision is the proportion of times that the model was correct when classifying something as positive.

Recall

$$Recall = \frac{true\_positives}{true\_positives + false\_negatives} = \frac{|gold \cap candidates|}{|gold|} \tag{2.2}$$

The recall is the proportion of positive values that were actually captured by the model.

F1 Score

$$F1\_Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.3}$$

The F1 score is the harmonic mean of precision and recall. This metric is widely used because it strikes a balance between the other two metrics, and drops to zero when either of them is zero.

Accuracy

$$Accuracy = \frac{true\_positives + true\_negatives}{all\_samples} = \frac{|correct\_guesses|}{|samples|} \tag{2.4}$$

Unbalanced corpus

The accuracy is the ratio of values correctly labeled by the classifier with respect to the whole test corpus. If the test corpus is highly **unbalanced** (i.e. there are too many examples of one class respect of the other), this metric tends to favor models that are more prone to assign labels of the majority class. Because of this, in a binary scenario (or with few classes) the F1 Score is usually preferred over the accuracy as metric.

All these metrics have values between 0 and 1, and could be expressed as percentages. In our example, the values for the metrics would be the following:

- Precision: $10/(10+2) = 0.833$ (83.3%)
- Recall: $10/(10+4) = 0.714$ (71.4%)
- F1 Score: $2 \cdot 0.833 \cdot 0.714/(0.833+0.714) = 0.769$ (76.9%)
- Accuracy: $(10+8)/(10+3+2+8) = 0.783$ (78.3%)

These metrics can easily be extended to problems with more than two classes. For example, we could try to classify tweets that have a positive (`P`), negative (`N`) or neutral (`Neu`) sentiment. In this case, we would define precision, recall and F1 scores per class, considering one class against all the rest for each case. Then

we could define averaged metrics for all the classes. The most frequent ones are the **macro averaged** metrics (Macro-Precision, Macro-Recall and Macro-F1), which assign equal weight to all classes in the test set. Another set of metrics are the **micro averaged** (Micro-Precision, Micro-Recall and Micro-F1), which assign a weight to each class relative to the proportion of elements of that class in the test set. Finally, the accuracy for a multi-class problem can be defined as easily as for a binary problem, considering the sum of all correct guesses for any class, divided by the total number of samples in the test set.

Macro Average

Micro Average

As we will see in section 5.2, these simple and widely used metrics can be extended to define more complex metrics for evaluating structured outputs such as the ones provided by our parsing processes.

2.6 Neural Networks

Artificial Neural Networks are a family of machine learning methods originally inspired by the observation of how biological systems learn using complex networks of interconnected neurons [Mitchell, 1997]. The first model of artificial neuron, outlined in figure 2.12, was presented in 1943 by McCulloch and Pitts. The idea, taken from biological neurons, is that an artificial neuron would receive a series of inputs x_i (stimuli) that would be multiplied by different weights w_i (corresponding to the excitatory of inhibitory capacity of the dendrites), the results would be summed up in the body of the neuron (summation of potentials) and if it exceeds certain threshold, the neuron would output a signal y (action potential in neurons).

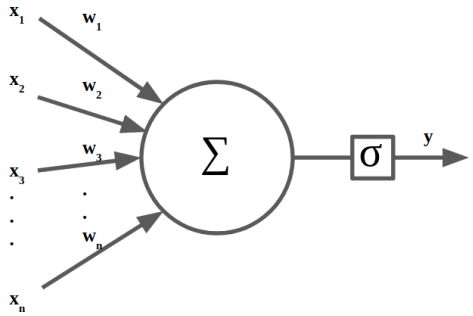


Figure 2.12: Simple artificial neuron.

As shown in the figure, the comparison against a fixed threshold is later on changed for a more expressive function σ, called the **activation function** of the neuron. So the equation of the output for this neuron would be:

Activation function

$$y = \sigma(\sum x_i \cdot w_i) \tag{2.5}$$

There are many possible activation functions, and generally they have some desirable properties like being differentiable and monotonic. Some examples of

usual activation functions include the logistic or sigmoid function $(1/(1+e^{-x}))$ or its multi-class generalization the **softmax** function, the hyperbolic tangent $(tanh(x))$ or the rectified linear unit (**relu**) function $(max\{0, x\})$.

Softmax

Relu

We can organize a collection of these neurons (also called units in the artificial neural network terminology) in an array where the inputs to all the units are the same x_i, and the corresponding output for each unit is y_j. This arrangement is called a **layer**, and it is one of the most common ways of organizing artificial neural networks (see section 2.6.2). In this case, we could use the notation $w_{i,j}$ for the ith weight corresponding to the unit in the jth output. We can then consider a matrix notation where the inputs are a vector $x = \{x_1 \dots x_n\}$, the outputs are a vector $y = \{y_1 \dots y_m\}$ and the weights are organized in a matrix $W = [[w_{i,j}]], i \in 1 \dots n, j \in 1 \dots m$. Then the equation for calculating the output of the whole layer can be written in this matrix notation using the product between x and W:

Layer

$$y = \sigma(xW) \tag{2.6}$$

Neural networks are a very flexible machine learning technique that can be used to solve a wide range of problems, in fact, they are universal approximators of functions [Cybenko, 1989; Hornik et al., 1989]. This is important from a theoretical point of view, and it helps us understand why they are so widely used, but they only started to be popular in the NLP community later on when the availability of data and computing power was enough to really put them to good use.

2.6.1 Training

Training a neural network is generally treated as an optimization process that tries to minimize a function related to the difference between the actual output of the network and its expected output. This function is called the **loss function**. There are many different loss functions used for different kinds of problems. Some of the most usual ones are the mean squared error loss (for regression problems) and the categorical cross entropy (for multi-class classification problems).

Loss function

Activation functions need to be differentiable because the training process will generally perform some form of gradient descent in order to minimize the loss. In particular, the networks can be trained using **stochastic gradient descent**, which implies starting with random weights and iteratively selecting a batch of samples, calculating the output for those samples, and adjusting the weights to make the actual output of the network more similar to the expected output, until some stop criteria is met.

Stochastic gradient descent

The calculation of the gradient used for updating the weights can be done efficiently using **backpropagation**. This means that, once the output for a sample has been calculated, it is possible to use the delta between the output and the expected value and the intermediate results for each unit inside the network layers to efficiently calculate the partial derivatives at each point and

Backpropagation

propagate this information back through the network in order to update all weights.

One possible stop criterion could be to iterate until the weights stabilize, or the difference between iterations is less than certain threshold. However, neural networks generally have a lot of parameters and thus tend to overfit very quickly to the training data. Because of this, it is usual to keep a held-out set of samples that are not used for training but for controlling the state of the network. The technique of **early stopping** is a criterion for stopping the training when the performance of the network over this held-out data has not improved after some iterations. This greatly improves the training time and generalization of some networks.

Early stopping

Another technique that is used to alleviate overfitting and improve generalization is **dropout**. In this technique, at each training step we randomly select a set of units that will not be used to calculate weights and will not be updated. Dropout is generally applied at layer level, setting for each layer the ratio of units that would be randomly selected not to be updated.

Dropout

2.6.2 Multilayer Perceptron

One of the simplest neural architecture architectures that is widely used is the Multilayer Perceptron (MLP). In essence, an MLP is a stack of layers of fully connected units (also called **dense layers**). The output of each layer is connected to the inputs of the following layer. For defininining an MLP, we need to set the number of layers, the number of units in each layer, and the activation function to use for each one of them.

Dense layer

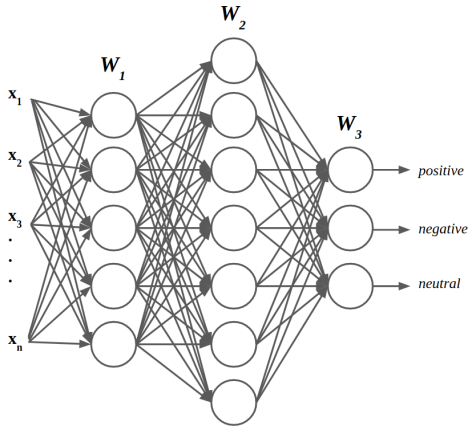


Figure 2.13: Example of an MLP with three layers.

For example, we could use a MLP architecture for the problem of predicting the sentiment of a tweet between positive, negative or neutral. Suppose we represent the tweet using bag of words, the input of the network would be a vector of the value for each word in the vocabulary. Figure 2.13 shows an

example of a MLP for this problem, using three layers with five, seven and three units each. In MLP terminology, the layers except the output layer are called hidden layers, so in this case we have a network with two hidden layers. Notice that the output of all units from each layer are connected to the inputs of the following layer. The activation function could be, for example, relu for each hidden layer, and softmax for the output layer, and we will consider the prediction of the network to be the unit with the highest output.

2.6.3 Long Short-Term Memory

Multilayer perceptrons are useful when the input has fixed size, but a great number of problems have representations with variable sizes. In the previous example, we modeled the tweet with a bag of words representation that has a fixed size. However, the bag of words representation has some shortcomings. For example, it does not take into account the relative order of words in a sentence, which is important information for many tasks. A more flexible representation would be having an input for each word, this could be done using a representation like **one-hot encoding**: the input vector has the size of the vocabulary, composed entirely of zeros except a value set to one in the position corresponding to the represented word.

One-hot encoding

We can create neural network architectures that consume a sequence of words in one-hot encoding, that are presented to the network one at a time. The Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) are two families of architectures that are aimed at this. Here we will discuss the Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] networks, which are a specific kind of Recurrent Neural Networks.

LSTM networks use a different kind of unit than the one presented in figure 2.12. First of all, they are recurrent in the way that the units have a loop that goes from the output and is injected back as an input. This means that the input of the unit at time $t+1$ will be a combination of some new inputs coming from upper layers plus the outputs of the same layer at time t. LSTM network units are composed of a cell, an input gate, an output gate and a forget gate. The cell is the memory of the unit, the value it stores, and at each step the output will be a combination of the value stored in this cell and the new input mediated by the forget gate.

When creating a LSTM layer containing several LSTM units, the layer is able to consume one input at each step, generate an output, and then use the inner state as partial input for the next step. During training, the network will tune how to use the forget gates so as to allow more information from the input or more information from the memory depending on the situation.

These layers can be used to output a value for each entry (for example to obtain a representation of each word of the sentence) or a final value after all the sequence has been consumed (for example to obtain a representation of the whole sentence). In our example, we could use a LSTM that consumes all the words of the tweet, one at a time, and outputs a representation with the most salient features of the tweet that could be used for calculating the sentiment

polarity. LSTMs are good at capturing relevant features in language such as long distance dependencies.

Bi-directional LSTM

There also exist variants of these networks. LSTMs can be used to consume the sequence from left to right or from right to left. Combining these, it is also possible to create **bi-directional LSTMs**: using a layer that reads the input from left to right and another layer that reads the input from right to left and then concatenating the outputs. We will be using these kinds of networks in some of the parsing architectures we will use (see section 5.5).

2.6.4 Deep Learning

The term deep learning is generally used to refer to the application of neural networks of many layers to solve different kinds of tasks. In the latest years, the use of neural networks in machine learning has been expanding rapidly and it has become the standard for many applications, including many tasks in NLP.

These advancements have been possible because of the amount of data, both labeled and unlabeled, that has been made readily available lately, and also improvements in computing power.

Although strictly we could say a multilayer perceptron could be considered a deep learning architecture because it contains many layers, the term is usually reserved to more complex architectures that involve one or more layers of other more complex types (such as CNNs or RNNs). The idea behind a deep learning architecture is that the first layers perform feature extraction transforming the input into low level features, while deeper layers use these high level features to extract higher level information. For example, in an image recognition task, the first layers might look at the input pixels to detect edges, deeper layers might identify shapes by combining different edges, and the final layers might recognize whole pictures from the basic shapes.

In NLP, the use of deep learning has been growing steadily since the 2000s, with works like: [Bengio et al., 2003], which presents a multi-layer neural network for computing a language model where one of the layers is used as a continuous vector representation of words (similar words tend to have vectors close in space); and [Collobert and Weston, 2008], which presents a unified neural network architecture that can perform several NLP tasks (including language modeling, POS-tagging, NER and SRL) using multi-task training and it can leverage the information learnt from the different tasks to improve the performance on others. These and other works paved the way for great advances in the application of neural networks to NLP tasks like the use of word embeddings and other more complex language modeling techniques.

2.6.5 Word Embeddings

Word embeddings

The notion of **word embeddings** refers to a collection of techniques for automated feature learning in the context of language modeling. It essentially means creating a representation for words in a dense vector space that is considerably smaller than the vocabulary size, hence embedding the language vocabulary into

another structure while preserving its most salient features. One key idea surrounding word embeddings comes from the notion of distributional semantics, that it is possible to infer the behavior of a word by looking at all the contexts it appears in.

The idea of word embeddings has been around for some time, but it took off mainly since the popularization of neural networks for NLP. Initial attempts like [Bengio et al., 2003] showed that when training a neural language model, the first layer acted as an embeddings layer for words. This layer could then be used to train other networks (like in [Collobert and Weston, 2008]) or even other types of classifiers (see [Ghosh et al., 2015; Rosá et al., 2017]). Over the years several techniques for creating embeddings collections were designed, involving architectures related to neural networks (such as [Mikolov et al., 2013; Bojanowski et al., 2017]) or other kinds of counting models (such as [Pennington et al., 2014]).

In particular, [Mikolov et al., 2013] proposed the very popular model known as `word2vec` for training word embeddings efficiently from large corpora. They presented two ways of casting the problem as a prediction problem (the Continuous Bag of Words and Continuous Skip-gram models) and propose a set of tests for measuring the quality of the generated embeddings based on the intuition that similar words would have similar vectors in space. In this work, we will use word embeddings based on the Continuous Bag of Words model (CBOW).

The CBOW model proposes a fake task for training word embeddings (see figure 2.14): considering a window of words, try to predict the word in the middle $w(t)$ given the surrounding words $\{w(t-n)\dots w(t-1), w(t+1)\dots w(t+n)\}$. The words in the window are represented using one-hot encoding with a vocabulary of size V, but they pass through a layer that outputs E units. This is the **embeddings layer**, a layer of size $V \cdot E$ that transforms the one-hot enconding input into a dense vector of E units for representing words. The aim of this fake task is in fact to train this layer, which could later on be used in other architectures as a pre-trained embeddings layer. The resulting vectors are summed up, returning another vector of E units, which goes through a softmax layer with V outputs, for predicting the word in the middle.

Embeddings layer

Using this fake task it is possible to efficiently train the embeddings layer using large corpus of text [Mikolov et al., 2013]. The resulting word embeddings have interesting characteristics, like the ability to capture some syntactic or semantic relationships between words (also called word analogies in the literature) using simple algebraic operations. For example, assuming $v(w)$ is the vector representation of word w, after training an embeddings collection (in English, Spanish, or other languages) it is usual that we can make some morphological correspondences like:

$$v(rey) - v(hombre) + v(mujer) \simeq v(reina)$$
$$v(king) - v(man) + v(woman) \simeq v(queen)$$

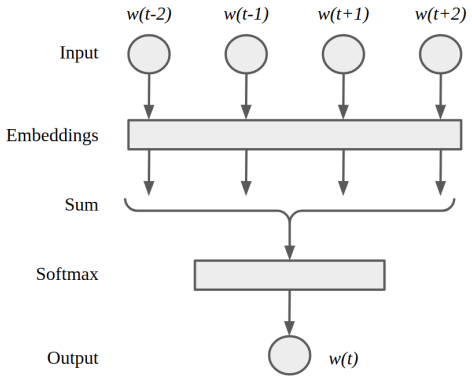


Figure 2.14: CBOW fake task architecture. In the diagram we consider a windows of size two (two words to the left and to words to the right) around the predicted word.

Also, we can make some semantic correspondences like:

$$v(parís) - v(francia) + v(uruguay) \simeq v(montevideo)$$
$$v(paris) - v(france) + v(uruguay) \simeq v(montevideo)$$

`word2vec` and other word embeddings techniques create vector representations at word level, which means a word will have a corresponding vector no matter the context in which it is used. In natural language, it is usual that words have ambiguous semantics, and depending on their context they might mean different things. So in the following years there have been attempts to create more powerful vector representations that can capture the meaning of words in context or even create vector representations of entire sequences of words. Two of these models are: ELMo [Peters et al., 2018] (Embeddings from Language MOdels), which represents the words as character-level convolutions and uses two layers of bi-directional LSTMs to encode the words in the sentence, allowing to obtain embeddings for a word in context or for the whole sentence; and BERT [Devlin et al., 2018] (Bidirectional Encoder Representations from Transformers), which uses the transformers [Vaswani et al., 2017] neural networks architecture. In this work, we will not use these more advanced models, and just use embeddings calculated using the `word2vec` technique.

2.6.6 Neural Parsing

Similarly to other NLP tasks, the performance in parsing has improved dramatically since neural methods started to be applied massively. Most of the work on applying neural network architectures to parsing has been done for the two more classical syntactic paradigms (constituency and dependency parsing), although there are some attempts at applying neural networks to deep parsing as well, mostly for English. In this section we will try to make a review of some of the most important works related to parsing using neural networks. This discussion will present in broad lines what the different neural parsers do, and we defer the performance comparison to the next section.

Constituency parsing

For constituency parsing, the work of [Socher et al., 2011a,b, 2012, 2013] is very relevant. They define a class of networks dubbed as Recursive Neural Networks, which perform a recursion over trees, instead of a recursion over sequences as a standard Recurrent Neural Network does. There are different variants of this architecture in the different works, but one of the key elements in the networks is trying to generate a binary constituency tree by looking at pairs of elements in a bottom-up fashion. Starting from the words of the sentence, the core network takes a pair of consecutive words as input, encoded as embeddings, and outputs another vector which represents an embedding of the combination of both words, and a score that represents if this pair of words should be combined as a constituent in the tree. They take the highest scoring pair and combine them, and proceed to repeat the process with the remaining elements. This is the greedy version of the process, a refinement of this is using a beam search to consdier several possibilities instead of the greedy process, making it more similar to a CKY. The different works present variants of this architecture growing in complexity, with the most complex one being the Recursive Neural Tensor Network, which they used to improve the performance of a sentiment analysis system.

A rather different but very interesting approach is followed by [Vinyals et al., 2015], which frames the parsing process as a sequence-to-sequence problem akin to machine translation. They try to learn a translation model where one of the languages is the collection of sentences in natural language, and the other language is the same collection of sentences but written as bracketed representations of parse trees. The neural model they use is an encoder-decoder LSTM model with attention mechanism. They found out that training such a model using only standard corpus data (training sections of the Penn Treebank) did not yield good results, but if they first pre-trained the model using a larger corpus of sub-standard data (the result of another parser, or a combination of parsers) the performance of the final model improved significantly.

The transition-based parsing algorithm (see appendix B.2) can also be applied to constituency parsing, considering the parsing process of a sentence as a sequence of actions that are performed to each word transversed from left to

right. Several works focus on creating neural based versions of this algorithm. For example, Dyer et al. [2016] describes a transition based constituency parser that uses neural networks for representing the state of the stack [Dyer et al., 2015], the sentence buffer and the current derivation history. With all this input, the network tries to predict the next action of the parser (shift, reduce, or adding a new non-terminal to the stack).

Liu and Zhang [2017] combine this approach with sequence-to-sequence modeling. They train a model that tries to translate an input sentence into a sequence of actions that a shift-reduce parser should take. They apply the same idea for a dependency parser [Nivre et al., 2007] and a constituency parser [Dyer et al., 2016], both for English, obtaining state of the art results.

Watanabe and Sumita [2015] propose a transition based approach for constituency parsing where they model the stack using a RNN or Elman Network, obtaining good results for English and Chinese treebanks, while [Cross and Huang, 2016] describes a transition based constituency parser that uses LSTMs for representing word spans instead of partially derived trees, obtaining good results for parsing English and French.

Another approach that focuses on determining the word spans in the tree is used in [Stern et al., 2017], which describes a top-down parser that calculates scores and constituency labels for each span in the input sentence, and uses a dynamic programming approach to find the partition of spans that generates the best possible tree. They also report that, instead of evaluating all spans, it is possible to greedily split the sentence in spans recursively in a top-down fashion without losing much performance. The spans are encoded using LSTMs in order to generate an intermediate representation that is used for calculating the score and the constituency labels. They present results for English and French. This model builds a binarized version of the trees, but a later refinement of the model presented in [Gaddy et al., 2018] is able to build trees with non-binary rules. They use CKY for finding the optimal tree, but introduce an empty label for indicating spans that are not constituents, so the parent constituent of these spans can be considered an n-ary rule.

Shen et al. [2018] describes a parser based on LSTMs trained to predict the syntactic distances (concept related to the distance between words in the expected parse tree) between consecutive words in the sentence. With the predicted syntactic distances, it is possible to build the tree in a top-down process splitting constituents guided by the relative syntactic distances between words. This process is faster as it only needs to calculate the syntactic distances for the sentence once. They achieve good results for English and Chinese parsing.

Dependency parsing

The transition-based dependency parsing algorithm (see appendix B.2) has been adapted using neural networks. For example Chen and Manning [2014] presents a greedy transition based parser where they input a representation of the parser configuration (words, parts of speech, arcs) to the neural network and try to predict the next action to take. Their training and evaluation data includes the

Stanford dependency conversion of Penn Treebank and other data.

The already mentioned work by [Liu and Zhang, 2017] performs both dependency and constituency parsing. This transition-based parser frames the problem as a sequence-to-sequence translation problem, trying to predict the sequence of actions given the sentence.

Other works for neural dependency parsing use the graph-based method instead of the transition-based method. Graph-based dependency parsers essentially assign a weight to every possible arc in the graph and find a maximum spanning tree using these weights. One of these works is [Dozat and Manning, 2017], where they use a biaffine attention mechanism for scoring the weights for every pair of words in the sentence.

Clark et al. [2018] also uses a graph-based method but uses multi-task for enriching the embeddings used for parsing. They train the dependency parsing model together with models for CCG supertagging, text chunking, NER, POS tagging and machine translation.

Zhang et al. [2020] is another example of neural graph-based dependency parsing, that extends the idea from [Dozat and Manning, 2017] but includes information from second order subtrees in the calculation of scores.

The works mentioned so far try to perform parsing for English. For Spanish, the interest in dependency parsing started to grow since its inclusion in the multi-lingual dependency parsing task at CoNLL-X [Buchholz and Marsi, 2006], however, neural networks were not widely used yet at that time. Spanish was also present in more recent editions of the task that focused on using the Universal Dependencies framework, like CoNLL 2017 [Zeman et al., 2017] and CoNLL 2018 [Zeman et al., 2018] shared tasks on multilingual parsing from raw text to Universal Dependencies. One of the baseline systems in CoNLL 2017, UDPipe [Straka and Straková, 2017] provides tokenization, morphological analysis and dependency parsing for several languages including Spanish, and uses a transition-based method implemented as a simple neural network with one hidden layer. The best performing parser for Spanish in CoNLL 2017 is described in [Dozat et al., 2017], and it is essentially the same parser as [Dozat and Manning, 2017]. For CoNLL 2018, [Che et al., 2018] presents a parser that uses an architecture similar to [Dozat and Manning, 2017] but includes contextualized embeddings (using ELMo) and also trains with treebanks from several languages to leverage the similarities between language families, resulting in high performance for many languages including Spanish.

Deep parsing

Most of the work over the last years applying neural network architectures to the parsing of deep formalisms has been done for the English language. Some of these works deal with improving the performance of supertagging, and then using other methods such as CKY for parsing. For example Xu et al. [2015] uses a recurrent neural network to improve supertagging accuracy for CCG, which improves parsing performance, while [Vaswani et al., 2016] improves it even further using a bi-LSTM architecture for the supertagger.

Other works try to model all the parsing process using neural networks, generally performing supertagging followed by another neural network approach to parsing. Ambati et al. [2016] does this for CCG, combining the supertagging approach with a transition based parser; while [Kasai et al., 2017; Friedman et al., 2017] follow a similar approach for TAG parsing.

For HPSG, the work by [Zhou and Zhao, 2019] is very relevant as they try to derive a HPSG grammar from the Penn Treebanks in English and Chinese and use a self-attention based mechanism followed by a CKY decoder to parse them, obtaining very good results. One interesting aspect of this work is that they use a dependency version of Penn Treebank to enrich the original constituents with head information, using this data to derive a bare HPSG structure. The attention mechanism scores the possible spans in the sentence and this information is used by the CKY decoder. They incorporate the use of embeddings from ELMo, BERT, and XLNet [Yang et al., 2019] in order to further improve their results.

This approach of simplified HPSG and its span representations was later used by [Mrini et al., 2019], modifying the attention mechanism so that each attention head represents a label (e.g. a syntactic category), they dub this the Label Attention Layer. Using this mechanism they improve the parsing performance over the English and Chinese Penn Treebanks.

2.7 Notes on the State of the Art

All the highest performing methods for parsing nowadays use neural network architectures. This is the case for constituency parsing, dependency parsing and also for parsing with deep grammar formalisms.

The definition of the parsing performance metrics used for each formalism will be discussed in section 5.2, but let us present here a quick overview of these metrics so as to present a comparison of results. Constituency parsing uses variants of precision, recall and F1 score but counting constituents in the sentences, both labeled and unlabeled (in this section we will present *U-F1* and *L-F1*). Dependency parsing, on the other hand, uses a kind of accuracy dubbed *attachment score*, also with labeled and unlabeled versions (*UAS* and *LAS*). There are not standard metrics that are used specifically for parsing in deep formalisms, and when comparing these types of systems they generally present constituency and dependency metrics, usually involving some kind of transformation to the output of the parsers (for example see [Clark and Hockenmaier, 2002]). However, as noticed in the literature [Ivanova et al., 2016], this has the effect of dampening the differences between parsers and their particular ways of modeling the language.

Most of the work has historically been done for the English language, and the comparison between parsers for that language is easier as all the parsers are evaluated at least against some of the variants of the Penn Treebank. Table 2.3 shows a comparison of parsing performance for several English parsers using different formalisms. All of them are evaluated against some version of the Penn

Treebank, but there might be some differences between them because not all the versions have exactly the same information. As well as this, not all works report the same metrics.

For comparison, we include two works for English previous to the neural networks popularization: Collins [1997] proposed a generative lexicalized parsing model for CFG, and Charniak [2000] described a maximum-entropy statistical model for CFG.

Parser	Formalism	U-F1	L-F1	UAS	LAS
[Collins, 1997]	CFG		87.8		
[Charniak, 2000]	CFG		89.5		
[Socher et al., 2011a]	CFG	90.29			
[Chen and Manning, 2014]	Dep			92.2	91.0
[Vinyals et al., 2015]	CFG		92.8		
[Watanabe and Sumita, 2015]	CFG		90.68		
[Xu et al., 2015]	CCG		87.04		
[Dyer et al., 2016]	CFG		92.4		
[Cross and Huang, 2016]	CFG		91.3		
[Vaswani et al., 2016]	CCG		88.32		
[Ambati et al., 2016]	CCG	90.09	83.33		
[Liu and Zhang, 2017]	CFG+Dep		93.4	93.1	90.1
[Stern et al., 2017]	CFG		91.79		
[Kasai et al., 2017]	TAG			90.97	89.68
[Friedman et al., 2017]	TAG			90.31	88.96
[Dozat and Manning, 2017]	Dep			95.7	94.1
[Gaddy et al., 2018]	CFG		92.08		
[Shen et al., 2018]	CFG		91.8		
[Clark et al., 2018]	Dep			96.6	95.0
[Zhou and Zhao, 2019]	HPSG		95.84	97.00	95.43
[Zhang et al., 2020]	Dep			96.14	94.49
[Mrini et al., 2019]	HPSG		96.38	97.42	96.26

Table 2.3: Performance of several English parsers evaluated over variants of the Penn Treebank. We show the main formalism used by each parser and their reported metrics.

For languages other than English, there is considerably less work on parsing and more disparity of resources. For Spanish different parsers have used different corpora for training and evaluation throughout the years, which makes it harder to have a fair and accurate comparison.

There are few works focusing on constituency parsing for Spanish, all from before the neural networks era. Cowan and Collins [2005] tries two approaches to improve standard PCFG parsers: including morphological information in the probabilistic model, and a reranking method with max-margin criterion trained over a set of global features from the parse trees. This second approach works best in their tests. Le Roux et al. [2012] experiments with Spanish parsing

using a PCFG with latent annotations with a simplified tagset. Both parsers are trained and evaluated over the Cast3LB corpus, which is a subset of the AnCora corpus.

As mentioned before, many initiatives in Spanish parsing were driven by its inclusion in CoNLL tasks beginning in CoNLL-X [Buchholz and Marsi, 2006]. The best approaches for Spanish in that edition were [McDonald et al., 2006] with a graph-based maximum spanning tree method, and [Nivre et al., 2006] a pseudo-projective dependency parser using Support Vector Machines. Lloberes et al. [2010] describes a dependency grammar and a rule-based dependency parser for Spanish (one of the Freeling parsers [Padró et al., 2010; Padró and Stanilovsky, 2012]) transforming the result of a shallow parser. They carry their experiments over the AnCora dependencies transformation used for CoNLL-X and also over the Sensem corpus [Castellón et al., 2006]. Later on Spanish was included in CoNLL 2017 [Zeman et al., 2017] and CoNLL 2018 [Zeman et al., 2018] as a new corpus in Universal Dependencies format, which is the format used by the latest parsers.

Table 2.4 shows the performance of the Spanish parsers described in this section and the previous one. Notice that not all parsers use the same evaluation corpus, and there are no works combining formalisms. As well as this, besides constituency and dependency grammars, there seem to be no works evaluating the performance of Spanish parsers on deep linguistic formalisms. We found only one work dealing with a TAG grammar for Spanish [Kolachina et al., 2011], but it only goes so far as to build the categories and a maximum entropy supertagger, not the full parsing process and evaluation.

Parser	Formalism	Corpus	L-F1	UAS	LAS
[Cowan and Collins, 2005]	CFG	Cast3LB	85.1		
[Nivre et al., 2006]	Dep	Ancora-DEP		84.67	81.29
[McDonald et al., 2006]	Dep	Ancora-DEP		86.1	82.3
[Lloberes et al., 2010]	Dep	Ancora-DEP		81.13	73.88
[Le Roux et al., 2012]	CFG	CastLB	85.47		
[Straka and Straková, 2017]	Dep	es-UD		87.91	84.95
[Dozat et al., 2017]	Dep	es-UD		87.29	
[Che et al., 2018]	Dep	es-UD		90.93	
[Zhang et al., 2020]	Dep	es-UD		90.86	

Table 2.4: Performance of several Spanish parsers for constituency and dependency grammars.

Chapter 3

Grammar

This chapter describes the grammar we use: a version of HPSG adapted to Spanish. In HPSG grammars, both lexical entries and rules are defined as feature structures, and their constraints (selection constrains or rule application constraints) are specified in their features. First of all we will introduce some fundamental concepts on HPSG and the notation we use throughout the document with a simple example of a parse tree. Then we will describe the general feature structure for expressions (both words and phrases) and give an overview of its features. Next we will present the different grammar rules that are used to combine the expressions and form the parse trees. Finally we will describe the principles we used in our grammar implementation.

3.1 Fundamentals and notation

Section 2.1.4 introduced some HPSG concepts in a light way, here we will begin to formalize the key ideas behind the grammar and present the notation we use to represent them in this document.

Remember the basic structure used in HPSG grammars is the feature structure, which is a collection of associations between features and values. We will represent a feature structure as a matrix written between squared brackets ('[' and ']') containing the corresponding features and their values. This is called an **attribute value matrix**. For example:

Attribute value matrix

$$\begin{bmatrix} \text{NUM s} \\ \text{GEN f} \end{bmatrix}$$

Features are noted using uppercase strings, values can be any arbitrary feature structure. The above example shows a simple grouping of two features and their values: feature `NUM` (used to model the number agreement feature of words) with value `s` (singular), and feature `GEN` (used to model the gender agreement feature of words) with value `f` (female). The example below shows a feature

structure that represents a marker for a noun `n` with an agreement feature `AGR`, which is itself another feature structure.

$$\begin{bmatrix} \text{n} \\ \text{AGR} \begin{bmatrix} \text{NUM} & \text{s} \end{bmatrix} \end{bmatrix}$$

It is possible to combine two feature structures using the unification operation (noted $\sqcup$), and the result will contain the features from both. If some feature is present in both structures, unification will proceed recursively and the result will have the unified values of the original structures.

$$\begin{bmatrix} \text{n} \\ \text{AGR} \begin{bmatrix} \text{NUM} & \text{s} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{n} \\ \text{AGR} \begin{bmatrix} \text{GEN} & \text{f} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{n} \\ \text{AGR} \begin{bmatrix} \text{NUM} & \text{s} \\ \text{GEN} & \text{f} \end{bmatrix} \end{bmatrix}$$

If two feature structures cannot unify (for example if they have different values for the same feature), we note this as $\perp$.

$$\begin{bmatrix} \text{n} \\ \text{AGR} \begin{bmatrix} \text{NUM} & \text{s} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{n} \\ \text{AGR} \begin{bmatrix} \text{NUM} & \text{p} \end{bmatrix} \end{bmatrix} = \perp$$

Expression

An HPSG analysis of a sentence is a tree where each node is either a word or a phrase. We will use the term **expression** to refer to either a phrase or a word. An expression is like a super-class of words and phrases.

We write lists of expressions between angled brackets ('<' and '>'):

- $< expression_1, expression_2, \cdots expression_n >$ represents a list of expressions.
- $< >$ represents an empty list.
- $< first \mid rest >$ represents a list composed by an expression $first$ and a list of expressions $rest$.

For defining an HPSG grammar, we need to specify the feature structure that is going to represent the expressions and the grammar rules. Rules are the definitions of the valid ways to combine expressions in order to create larger expressions. In our case, the rules are all going to be binary, and each one of them will define the daughter that is the syntactic head.

As mentioned in section 2.1.4, when describing a rule we write on the left the elements being combined (the daughters), an arrow ($\rightarrow$), and the result on the right (the parent structure), as shown in figure 3.1. Because our rules are always binary, on the left we will have the two daughters being combined separated by a plus (+) sign. The syntactic head is signaled with a '(H)' marker.

(H) *head_expression* + *dependent_expression* → *result_expression*

(a) Left-headed rule.

dependent_expression + (H) *head_expression* → *result_expression*

(b) Right-headed rule.

Figure 3.1: Notation for rules.

As we will see, most of the rules we use have two variants, for example you can apply a complement or a subject to the left or to the right of a head. Although Spanish is nominally a SVO language like English, sentences with different word order are very common and in many cases sound more natural than their SVO counterparts (see for example [Di Tullio and Malcuori, 2012]). Because of this, for most of the rules we have left-headed of right-headed variants, depending on which daughter they define as their syntactic head. The figure shows the notation for the two rule flavors: the left-headed rules (figure 3.1a) and the right-headed rules (figure 3.1b).

Rules are implemented as feature structures as well, so this simplified notation for rules is only syntactic sugar. See section 3.11 for details about how we implement these rules as feature structures.

A box with a number like $\boxed{1}$ is a coindexation marker. They represent that the positions indicated by the markers with the same number in the structure have the same value. Let us see, for example, how to define `head_comp` rule, which attaches a head to a complement on its right (see section 3.4):

$$(\text{H})\begin{bmatrix}\text{expr} \\ \text{VAL} \quad \left[\text{COMP} \quad \left\langle \boxed{1} \mid \boxed{2} \right\rangle\right]\end{bmatrix} + \boxed{1} \rightarrow \begin{bmatrix}\text{phrase} \\ \text{VAL} \quad \left[\text{COMP} \quad \boxed{2}\right]\end{bmatrix}$$

Notice this is a left-headed rule because the '(H)' marker is on the left expression. The left expression has a `COMP` feature defined as a list with first expression $\boxed{1}$ and rest $\boxed{2}$. The first expression is coindexed with the daughter on the right (only represented as the marker $\boxed{1}$), while the rest is coindexed with the `COMP` feature of the resulting expression.

Finally, let us put these concepts together with a simple example of an HPSG analysis. Consider the following sentence:

(20) *Juan come manzanas rojas (John eats red apples)*

Using our grammar, sentence 20 would be analyzed as the tree in figure 3.2. Notice that the representation of the sentence is a tree where the leaves are words and the internal nodes are phrases. In the figure, word nodes are labeled with their corresponding words, while phrase nodes are labeled with their corresponding rule.

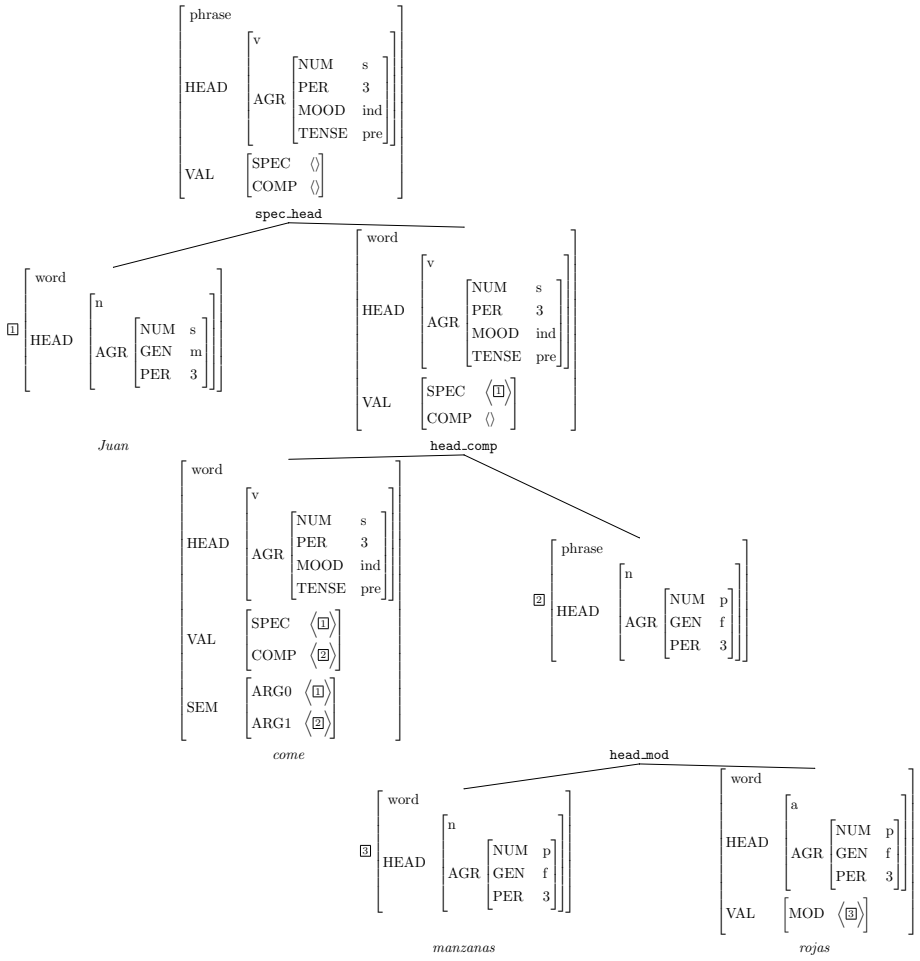


Figure 3.2: Analysis of the sentence *"Juan come manzanas rojas"* (*"John eats red apples"*).

Notice that each expression has a `HEAD` feature that defines its part of speech and contains its morphological attributes (in the agreement `AGR` feature). Furthermore, the expressions define other features that indicate how they can be combined (in the valence `VAL` feature). Section 3.2 describes all the features we use for expressions.

This example first applies the modifier rule `head_mod` to combine *"manzanas"* and *"rojas"*; then the result is applied as a complement of *"come"* using the complement rule `head_comp`; and finally the subject *"Juan"* is applied as a subject to the structure using the specifier rule `spec_head`. Each rule imposes constraints on the features the expressions must have in order to combine them, and the expressions themselves might define further constraints.

In HPSG the parse trees are not simple structures but reentrant trees, i.e., there can be more links between a pair of nodes than the parent-daughter relation. This is represented in the tree by the coindexation markers 1, 2 and 3. For example, the object *"manzanas rojas"*, marked as 2 is set as the value for the features `COMP` and `ARG1` of the verb *"come"*.

In our case, the difference in terms of features between words and phrases is that words might include a `SEM` feature for defining the semantic structure of the word. Notice that the verb *"come"* in this case defines the features `ARG0` and `ARG1`, which are coindexed with other values in the structure (the subject and the complement). This means that the subject of the verb (*"Juan"*) corresponds to the agent of the predicate, and the object (*"manzanas rojas"*) corresponds to the theme. These values might be switched in a passive voice construction, and in that case the feature structure for the verb would reflect this difference.

In the following sections we will define what are the possible features for each structure, and how each one of the rules work.

3.2 Feature structure for expressions

The feature structure we use for a word or phrase contains the features shown in figure 3.3. Notice that many these features are implemented as lists of expressions but in some cases the only possibilities for the list are only zero or one expressions.

$$\left[\begin{array}{ll}
\text{word} & \\
\text{HEAD} & \left[\begin{array}{ll} \text{n} \mid \text{v} \mid \text{a} \mid \ldots & \\ \text{AGR} & \left[\begin{array}{ll} \text{PER} & 1 \mid 2 \mid 3 \\ \text{NUM} & \text{sg} \mid \text{pl} \\ \text{GEN} & \text{f} \mid \text{m} \\ \ldots & \end{array}\right] \end{array}\right] \\
\text{VAL} & \left[\begin{array}{ll} \text{SPEC} & \langle \text{expr} \rangle \\ \text{COMP} & \langle \text{expr} \rangle \\ \text{CLITIC} & \langle \text{expr} \rangle \\ \text{MOD} & \langle \text{expr} \rangle \\ \text{LEFT_COORD} & \langle \text{expr} \rangle \\ \text{RIGHT_COORD} & \langle \text{expr} \rangle \end{array}\right] \\
\text{NONLOCAL} & \left[\text{REL} \quad \langle \text{expr} \rangle\right] \\
\text{SEM} & \left[\begin{array}{ll} \text{ARG0} & \langle \text{expr} \rangle \\ \text{ARG1} & \langle \text{expr} \rangle \\ \ldots & \\ \text{IS_ARG0} & \langle \text{expr} \rangle \\ \text{IS_ARG1} & \langle \text{expr} \rangle \\ \ldots & \end{array}\right] \\
\text{TEXT} & \text{text}
\end{array}\right]$$

Figure 3.3: Feature structure for a word.

Syntactic features

- `SPEC`: List of expressions that are expected as specifiers of a node. In general there should only be zero or one of these. As in [Sag et al., 2003], the specifier feature is used to model both the specifiers of nouns (generally determiners) and the subjects of verbs.
- `COMP`: List of expressions that are expected as complements of a node. This list is unbounded, as some words (especially verbs) can have multiple complements, though in general it has up to four or five elements.

- MOD: List of expressions that this node is expected to modify. There can be only zero or one elements in this list.
- CLITIC: List of expressions that are expected as clitics of a node. Because of Spanish constraints, this list should have up to two elements.
- LEFT_COORD (and RIGHT_COORD): List of expressions this node is expected to have on its left (or right) in order to form a coordination (only zero or one expressions).
- REL: List of expressions that this node is expected to be modifying as a relative clause. There can be only zero or one elements in this list.

Figure 3.3 shows where these features are located inside the feature structure. Notice that the SPEC, COMP, MOD, CLITIC, LEFT_COORD, and RIGHT_COORD features are valence features, and are located inside a bigger feature called VAL. However, the REL feature is the sole feature inside the NONLOCAL structure, this is because relative clauses that act as modifiers are the only type of non-local structure we are modelling in this grammar, as explained in section 3.8. As we will see, SPEC, COMP and CLITIC are endocentric features (they are defined by a head that expects some dependents), while MOD, REL and the coordination features are used in an exocentric way (they are defined by a dependent that will be attached to a head).

Semantic features Inside the SEM feature, there are features for determining the semantic role label structure of the expressions that are attached to a word. Unlike the original HPSG grammars, we use a representation based on the PropBank [Bonial et al., 2010] notation of semantic arguments. The features we use are the following:

- ARG0, ARG1, ARG2, ARG3, ARG4, ARGM and ARGL: The PropBank features that are included as annotations in AnCora. ARG0 indicates a proto-agent, ARG1 indicates a proto-patient, while the rest of the numeric arguments indicate roles that depend on the verb. ARGM corresponds to adjuncts and ARGL indicates arguments of light verbs [Taulé et al., 2008].
- ARGC: This is a special feature we use to model verbal complements for verb phrases, as explained in section 3.5.
- IS_ARG0, IS_ARG1, IS_ARG2, IS_ARG3, IS_ARG4, IS_ARGM and IS_ARGL: These are the symmetric features for modeling arguments that are introduced by modifiers, and are thus exocentric.

3.3 Specifier rules

The two rules for attaching a specifier to the left or to the right of a head are called `spec_head` and `head_spec`. The definitions of the two rules are shown in figure 3.4: given a head that expects a `SPEC` and a dependent, the rule unifies the dependent with the `SPEC` feature and returns a phrase with the `SPEC` feature saturated (empty). The head expression will also specify other selection constrains, e.g. a verb head might indicate that the expected `SPEC` has to be a noun.

$$\boxed{1} + (\text{H}) \begin{bmatrix} \text{expr} \\ \text{VAL} \quad \left[\text{SPEC} \quad \left\langle \boxed{1} \right\rangle\right] \end{bmatrix} \rightarrow \begin{bmatrix} \text{phrase} \\ \text{VAL} \quad \left[\text{SPEC} \quad \langle\rangle\right] \end{bmatrix}$$

(a) `spec_head`

$$(\text{H}) \begin{bmatrix} \text{expr} \\ \text{VAL} \quad \left[\text{SPEC} \quad \left\langle \boxed{1} \right\rangle\right] \end{bmatrix} + \boxed{1} \rightarrow \begin{bmatrix} \text{phrase} \\ \text{VAL} \quad \left[\text{SPEC} \quad \langle\rangle\right] \end{bmatrix}$$

(b) `head_spec`

Figure 3.4: The left-headed and right-headed versions of the specifier rule.

The most common version of this rule is `spec_head`, shown in figure 3.4a. This is used for applying a determiner as specifier of a noun, or a subject to the left of a verb. The `head_spec` rule is used in the cases of postponed subject (*sujeto pospuesto*), a very common phenomenon in Spanish.

Figure 3.5 shows the analysis of the noun phrase *"las manzanas" ("the apples")*, as a typical example of a determiner specifier applied to a noun. Other typical cases involve other parts of speech, such as a number (`z`) acting as the specifier of a noun.

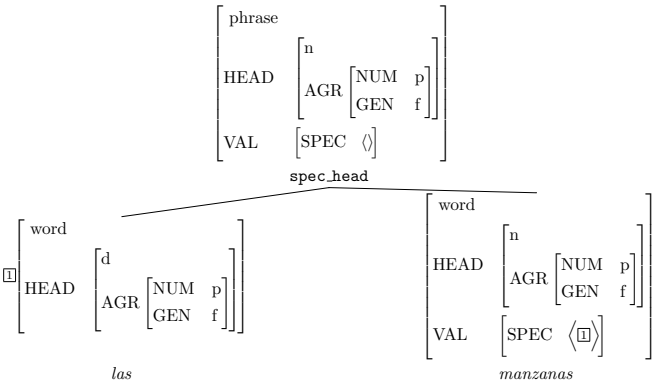


Figure 3.5: Analysis of the noun phrase *"las manzanas" ("the apples")*.

Figures 3.6 and 3.7 show two variants of the sentence *"the train arrived"* in Spanish: one of them using the subject on the left like in English (*"el tren llegó"*), and the other equally valid using a postponed subject (*"llegó el tren"*). In both cases the result of the analysis is a sentence (a verb phrase that is not expecting any other argument). The analyses in both cases are very similar, but they use the right-headed or left-headed variants of the specifier rule, `spec_head` or `head_spec` respectively.

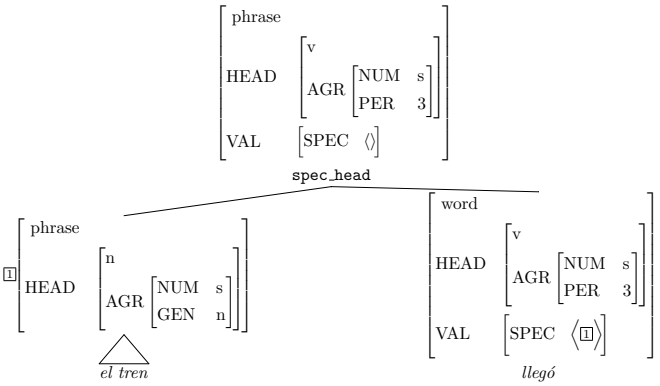


Figure 3.6: Analysis of the sentence *"el tren llegó" ("the train arrived").*

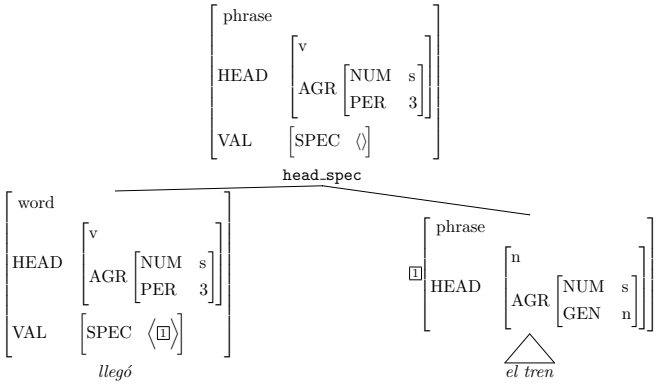


Figure 3.7: Analysis of the sentence *"llegó el tren"* (variant of *"the train arrived"*).

One potential problem that could arise from having both rules is that it could be possible to allow wrong analyses when applying this rule for non-verbal cases, for example the phrase **"manzanas las" ("apples the")*. However, as the aim is to create a statistical parser, we will in principle not rule out any combination and trust that the statistical process will learn such combinations are not possible as it will not see examples of those constructions in the training corpus.

3.4 Complement rules

The two rules used for attaching a complement to the left or to the right of a head are called `comp_head` and `head_comp`. The definition of the two rules are shown in figure 3.8. Similarly to what happens to subjects of verbs in Spanish, complements are usually located to the right of a verb, but occasionally they might be on the left.

$$\boxed{1} \; + \; (\mathrm{H}) \begin{bmatrix} \text{expr} \\ \text{VAL} \quad \left[\text{COMP} \quad \left\langle \boxed{1} \mid \boxed{2} \right\rangle\right] \end{bmatrix} \rightarrow \begin{bmatrix} \text{phrase} \\ \text{VAL} \quad \left[\text{COMP} \quad \boxed{2}\right] \end{bmatrix}$$

(a) `comp_head`

$$(\mathrm{H}) \begin{bmatrix} \text{expr} \\ \text{VAL} \quad \left[\text{COMP} \quad \left\langle \boxed{1} \mid \boxed{2} \right\rangle\right] \end{bmatrix} + \; \boxed{1} \rightarrow \begin{bmatrix} \text{phrase} \\ \text{VAL} \quad \left[\text{COMP} \quad \boxed{2}\right] \end{bmatrix}$$

(b) `head_comp`

Figure 3.8: The left-headed and right-headed versions of the complement rule.

Figures 3.9 and 3.10 show two variants of the verb phrase *"said something"* in Spanish: one of them using the complement on the right like in English (*"dijo algo"*), and the other variant uses the complement on the left (*"algo dijo"*). In both cases the result of the analysis is a verb phrase that expects a subject to form a sentence. The analyses in both cases are very similar, but they use the left-headed or right-headed variants of the complement rule, `head_comp` or `comp_head` respectively. Furthermore, as we will see in section 3.8, the use of the right-headed rule `comp_head` will be specially important for the analysis of relative clauses.

The left-headed `head_comp` is also used to analyse prepositional phrases such as *"de madera"* (*"wooden)"* or *"en casa"* (*"at home"*). The prepositions define through the `COMP` feature the type of expression they are expecting.

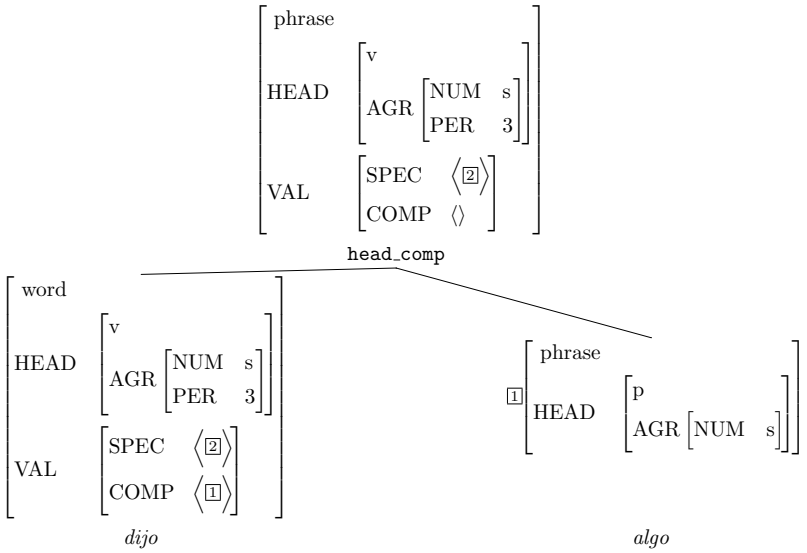


Figure 3.9: Analysis of the verb phrase *"dijo algo"* (*"said something"*).

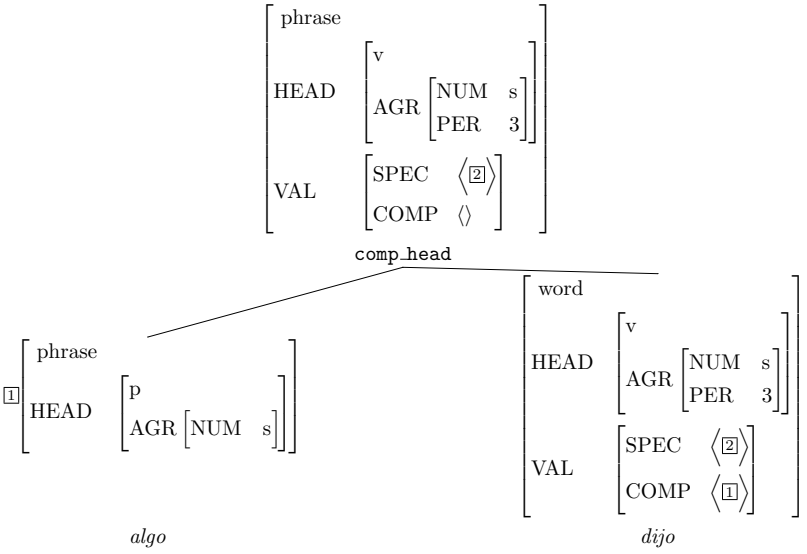


Figure 3.10: Analysis of the verb phrase *"algo dijo" (variant of "said something").*

3.5 Semantic complement rule

Consider the example sentences 21 through 26 that use the Spanish verbal form *pretérito pluscuamperfecto* (roughly equivalent to a past perfect in English), which could all be translated as *"John had said something"*.

(21) *Juan había dicho algo*

(22) *Algo había dicho Juan*

(23) *Juan algo había dicho*

(24) *Había dicho algo Juan*

(25) *Había dicho Juan algo*

(26) *Algo Juan había dicho*

(27) * *Juan dicho había algo*

This structure is always formed by the auxiliary verb *"haber"* and a past participle, where the auxiliary verb is inflected in order to match the subject. Notice that sentences 21 through 26 are all variants of the same sentence but moving the subject and object to the beginning or to the end. However, sentence 27 is ungrammatical, consisting of the participle preceding the auxiliary verb. It seems to be that once the verb phrase *"había dicho"* is built, it can be moved around the sentence as a unit, expecting the subject and the object in any position, but it cannot be built in an incorrect order.

We model this behavior in our grammar with a rule for building this kind of verb phrases that act as a unit. We want the verb phrases to have the following properties:

Support verb

- There is a conjugated verb, whose number and person will agree with the subject. We will call it the **support verb**[1].

Content verb

- There is another verb in a non-finite form (in this case a participle) that carries the main semantic content for the verb phrase. We will call it the **content verb**.
- The only arguments that the support verb expects are the subject and the content verb.
- All the other arguments around the verb phrase should be applied to the content verb, instead of the support verb.

It is as if these verb phrases have two different heads: a syntactic head (in this case *"había"*) which carries the agreement features) and a semantic head (in this case *"dicho"*) which carries the argument valence features. In order to analyze these cases, we use a variant of the complement rule that works like `head_comp` but percolates the dependent valence arguments to the resulting phrase. This is the `head_comp_sem` rule, as shown in figure 3.11.

[1] In this work, we are using the expression *support verb* with a different meaning than the usual *verbo soporte* in Spanish [Ingelmo, 2002].

$$\text{(H)}\begin{bmatrix}\text{expr} & \\ \text{HEAD} & \boxed{2} \\ \text{VAL} & \left[\text{COMP}\left\langle \boxed{1} \right\rangle\right]\end{bmatrix} + \boxed{1}\begin{bmatrix}\text{expr} & \\ \text{VAL} & \boxed{3}\end{bmatrix} \quad \rightarrow \begin{bmatrix}\text{expr} & \\ \text{HEAD} & \boxed{2} \\ \text{VAL} & \boxed{3}\end{bmatrix}$$

Figure 3.11: The `head_comp_sem` rule.

Figure 3.12 shows the parse tree for sample sentence 22: *"Algo había dicho Juan""*. Notice that the support verb *"había"* uses the `ARGC` semantic feature to mark its verbal complement, and that the content verb *"dicho"* could correctly fulfill both of its semantic arguments `ARG0` and `ARG1`.

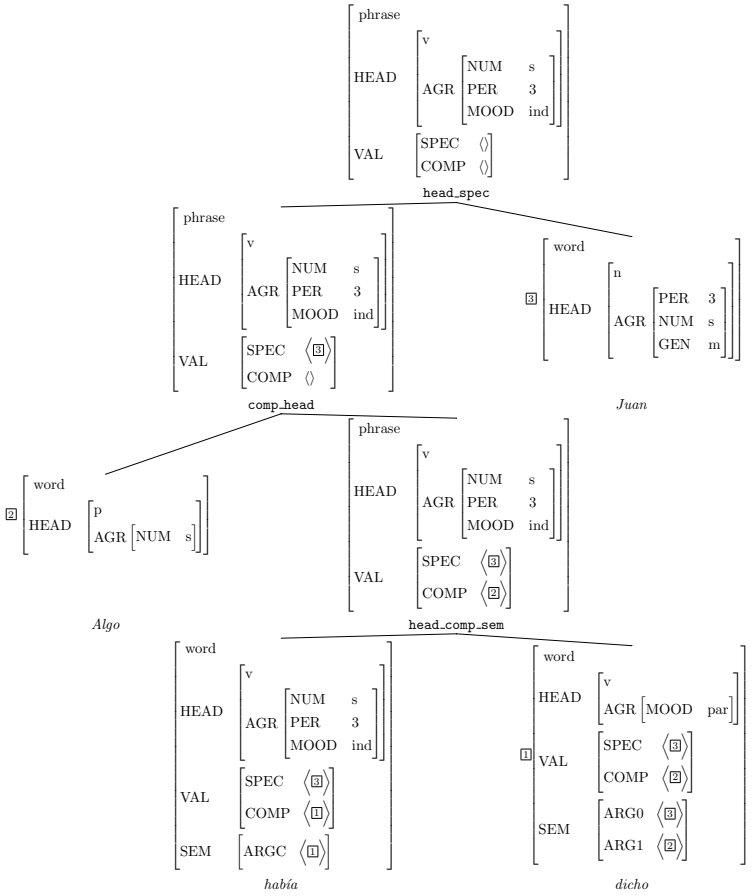


Figure 3.12: Analysis of the sentence *"Algo había dicho Juan"* (*"John had said something"*).

Verb phrases built with the auxiliary verb *"haber"* are not the only structures in Spanish that could be analyzed in this way. Something similar happens in example 28, though in this case the verb phrase does not use an auxiliary verb but a modal, we still would like to associate the object to the main lexical verb *"decir"* and the subject should agree with the inflected verb (the syntactic head) *"podía"*. Other verb phrases that could use the same analysis are the ones shown in examples 29 and 30. These constructions can also be nested, such as in 31.

(28) *Juan podía decir algo (John could say something)*

(29) *Juan tiene que ir a la escuela (John has to go to school)*

(30) *Juan comenzó a cantar una canción (John began to sing a song)*

(31) *Juan había comenzado a cantar una canción (John had began to sing a song)*

The `head_comp_sem` rule is flexible enough to model all these structures:

- Sentence 28 is analogous to 21 but using the support verb *"podía"* and the content verb *"decir"*.
- For sentence 29, we first apply `head_comp_sem` between *"que"* and *"ir"*, which creates the intermediate phrase *"que ir"* that expects the same arguments as *"ir"*, and finally we apply *"tiene"* to the phrase *"que ir"* using `head_comp_sem` again.
- The phrase *"comenzó a cantar"* in sentence 30 is analyzed in the same way as in sentence 29.
- For sentence 31 we can add one more nesting level, the final structure links *"había"* to *"comenzado"* and *"comenzado"* to *"a cantar"*, applies the remaining arguments to the content verb *"cantar"* and also coindexes *"Juan"* to the subject of *"había"*.

3.6 Modifier rules

Modifiers are structures that depend on a head but are not selected by it, i.e., they are not arguments that the head expects. Instead, modifiers can be attached more freely, and they define using the `MOD` feature which kind of structure they want to be attached to. Modifiers can also be applied in any order to the left or to the right of a head. The two rules for attaching a modifier to the left or to the right of a head are called `mod_head` and `head_mod`. The definitions of these two rules are shown in figure 3.13.

$$\begin{bmatrix}\text{expr}\\ \text{VAL} & \left[\text{MOD} & \left\langle \boxed{1} \right\rangle\right]\end{bmatrix}+ \quad (\text{H})\,\boxed{1} \rightarrow \boxed{1}$$

(a) `mod_head`

$$(\text{H})\,\boxed{1} \; + \; \begin{bmatrix}\text{expr}\\ \text{VAL} & \left[\text{MOD} & \left\langle \boxed{1} \right\rangle\right]\end{bmatrix} \rightarrow \boxed{1}$$

(b) `head_mod`

Figure 3.13: The left-headed and right-headed versions of the modifier rule.

Figure 3.14 shows an example of application of the rule `head_mod` in the phrase *"perro negro"* (*"black dog"*). The adjective *"negro"* defines in its feature `MOD` that is expecting to modify a noun, which will be unified with the head *"perro"*. Notice that the resulting phrase still carries other valence features that are still pending from the noun (in this case the `SPEC` feature).

$$\begin{bmatrix}\text{phrase}\\ \text{HEAD} & \begin{bmatrix}\text{n}\\ \text{AGR} & \begin{bmatrix}\text{NUM} & \text{s}\\ \text{GEN} & \text{m}\end{bmatrix}\end{bmatrix}\\ \text{VAL} & \left[\text{SPEC} & \left\langle \boxed{2} \right\rangle\right]\end{bmatrix}$$

`head_mod`

$$\boxed{1}\begin{bmatrix}\text{word}\\ \text{HEAD} & \begin{bmatrix}\text{n}\\ \text{AGR} & \begin{bmatrix}\text{NUM} & \text{s}\\ \text{GEN} & \text{m}\end{bmatrix}\end{bmatrix}\\ \text{VAL} & \left[\text{SPEC} & \left\langle \boxed{2} \right\rangle\right]\end{bmatrix} \qquad \begin{bmatrix}\text{phrase}\\ \text{HEAD} & \begin{bmatrix}\text{a}\\ \text{AGR} & \begin{bmatrix}\text{NUM} & \text{s}\\ \text{GEN} & \text{m}\end{bmatrix}\end{bmatrix}\\ \text{VAL} & \left[\text{MOD} & \left\langle \boxed{1} \right\rangle\right]\end{bmatrix}$$

perro *negro*

Figure 3.14: Analysis of the noun phrase *"perro negro"* (*"black dog"*).

Figure 3.15 shows an example of application of the rule `mod_head` for the sentence *"ayer llovió"* (*"yesterday it rained"*). In this case, the adverb *"ayer"* must define in its feature `MOD` that it expects to modify a verb.

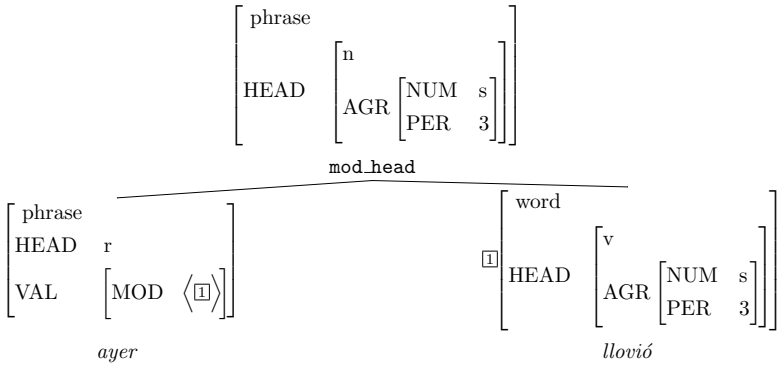


Figure 3.15: Analysis of the sentence *"ayer llovió"* (*"yesterday it rained"*).

3.7 Clitics rule

Clitic reduplication

Clitic pronouns in Spanish are pronouns that can appear before or after a verb and can either take the place of a verbal argument or be used together with the argument (this is called **clitic reduplication** or **clitic doubling**). If the clitic is used before the verb, it is written as a separate word, e.g. *"yo **lo** vi"* (*"I saw him"*) or *"yo **lo** vi a Juan"* (*"I saw Juan"*). On the other hand, when the clitic is used after the verb, it is conjoined with the verb (this is called *enclitic*), e.g. *"iba a ver**lo**"* (*"I was going to see him"*).

In our grammar, we only model the clitics that appear before the verb. We are not splitting conjoined words as they appear in the original AnCora corpus, so we do not model the enclitic case. Because of this, there is only one rule for clitics in the grammar that applies a clitic to the left of a phrase. The rule is called `clitic_head`, and it is shown in figure 3.16.

$$\boxed{1} \; + \; (\mathrm{H}) \begin{bmatrix} \text{expr} \\ \text{VAL} \quad \left[\text{CLITIC} \quad \left\langle \boxed{1} \mid \boxed{2} \right\rangle\right] \end{bmatrix} \rightarrow \begin{bmatrix} \text{phrase} \\ \text{VAL} \quad \left[\text{CLITIC} \quad \boxed{2}\right] \end{bmatrix}$$

Figure 3.16: The clitics rule `clitic_head`.

Figure 3.17 shows the analysis of the sentence *"Yo lo vi"* (*"I saw him"*) using the clitics rule. Notice that the clitic acts as `ARG1` of the verb. Clitics in Spanish often (though not always) act as semantic arguments of the verb, and in these cases they might take the place of `ARG1` or `ARG2`.

Compare this to figure 3.18, that shows the analysis of the sentence *"Yo lo vi a Juan"* (*"I saw John"*). In this case both the clitic pronoun *"lo"* and the prepositional phrase *"a Juan"* are set as `ARG1` of the verb. This is a case of clitic reduplication, which is a very common phenomenon in Spanish [Pineda and Meza, 2005]. Notice that in this same example, the clitic is applied to the head (verb) before the complement. We consider there is an implicit precedence of the clitic application over the combination with any other expression. However, this characteristic is not explicitly encoded in the grammar: the statistical process is expected to discover this property based on the training data.

In Spanish it is possible to apply up to two clitics for a verb. If both the indirect object and direct object clitics are present, the leftmost one must be the indirect object clitic. Figure 3.19 shows the analysis of *"Juan me lo dijo"* (*"John told me"*), which has indirect and direct object clitics, coindexed to `ARG2` and `ARG1` respectively.

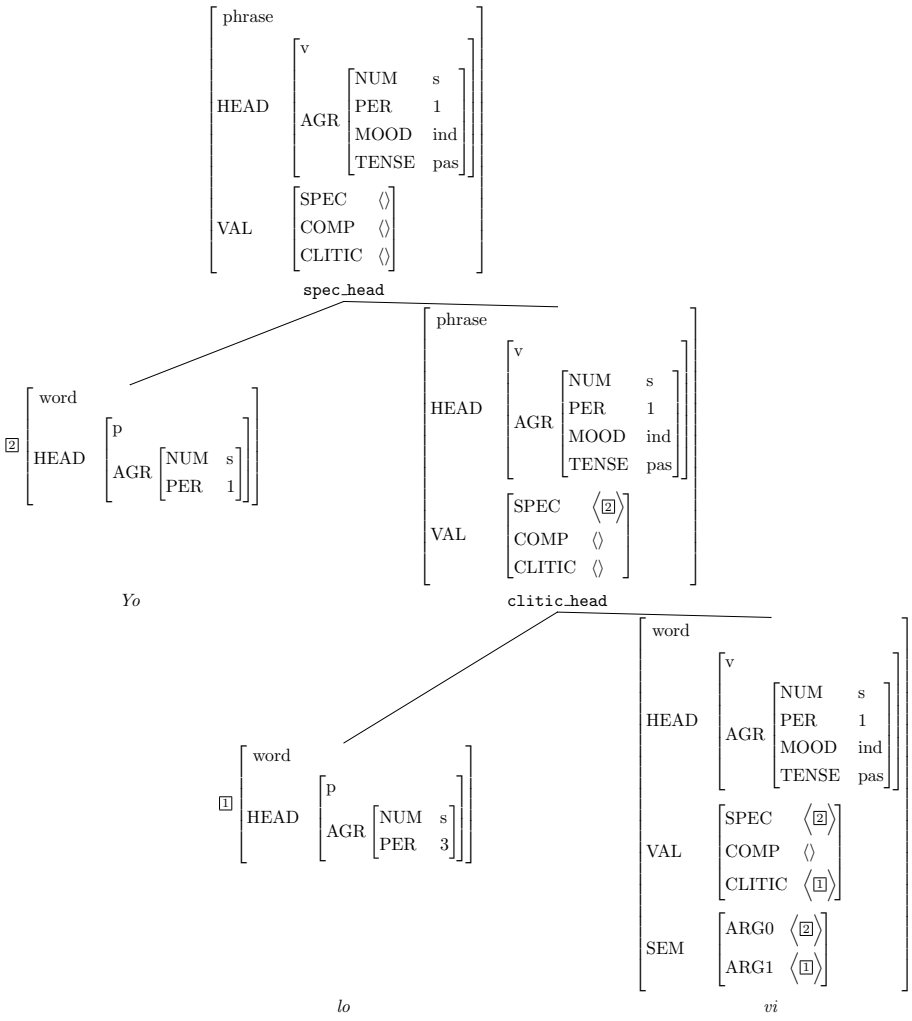


Figure 3.17: Analysis of the sentence *"Yo lo vi"* (*"I saw him"*).

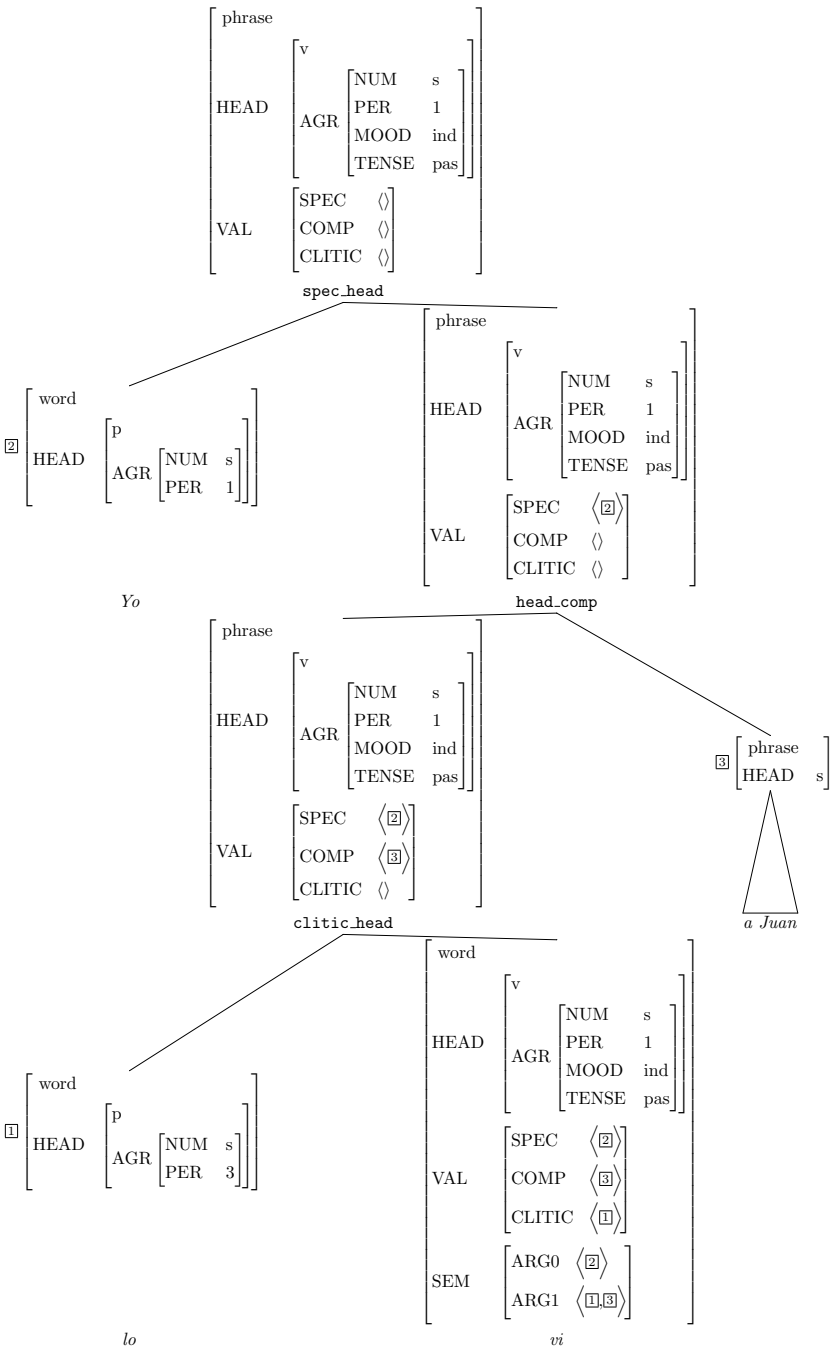


Figure 3.18: Analysis of the sentence *"Yo lo vi a Juan"* (*"I saw Juan"*).

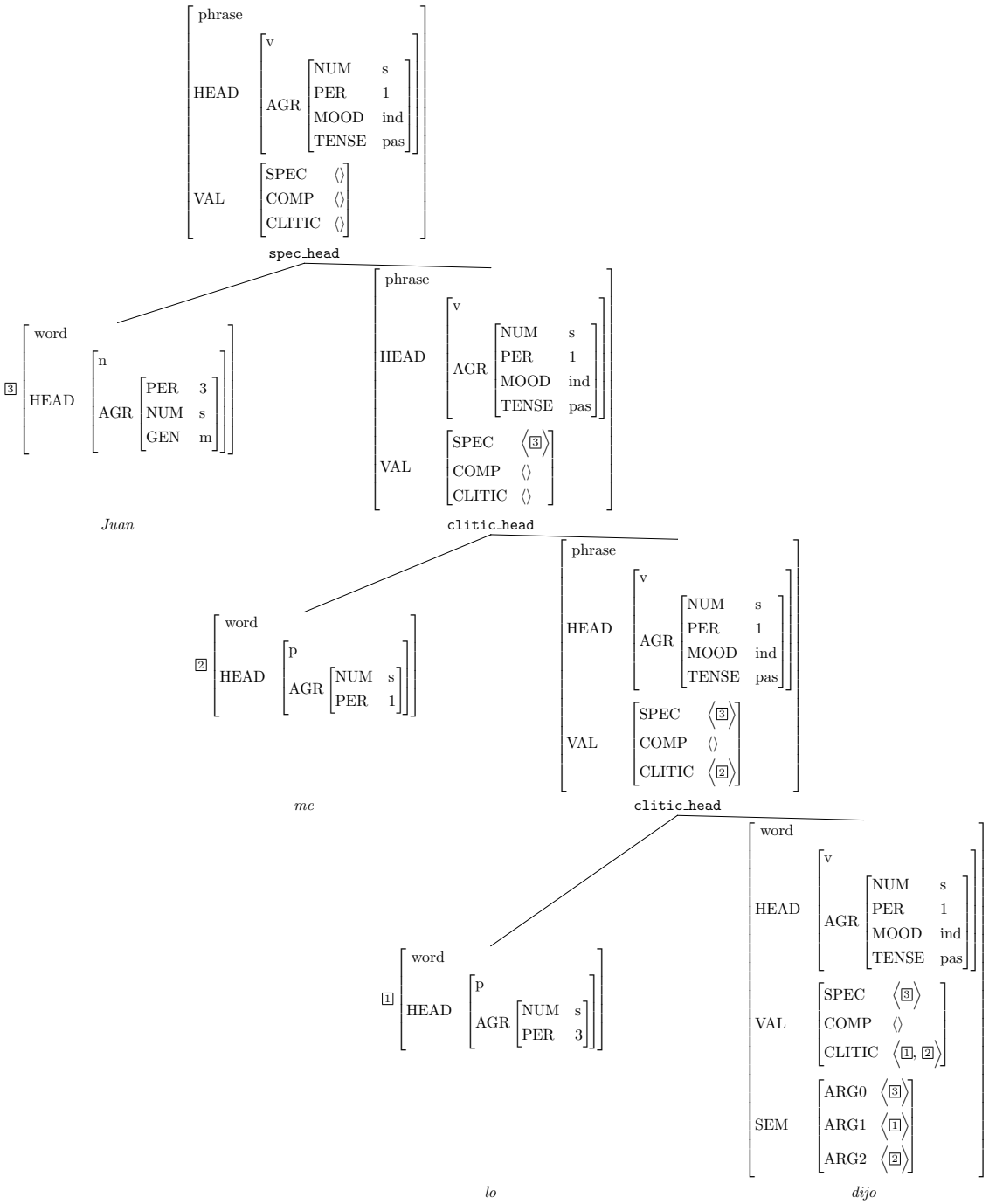


Figure 3.19: Analysis of the sentence *"Juan me lo dijo"* (*"John told me"*).

3.8 Relatives rule

Relative pronouns are pronouns that allow the introduction of a relative clause inside another sentence. Consider the following examples:

(32) *El libro que Juan leyó era largo / The book that John read was long*

(33) *El perro que me mordió era gris / The dog that bit me was gray*

The pronoun *"que"* (*"that"*) in sentences 32 and 33 is used to introduce the relative clauses. Notice that in these cases, the clauses *"que Juan leyó"* and *"que me mordió"* are acting as modifiers of the corresponding nouns, but at the same time the nouns play a role in the subordinate sentence (the book is the read object, and the dog is the one who bit me), even if they are syntactically out of the scope of the relative sentences. The relative pronouns are the links that allow to introduce a sentence with a missing dependency (a gap) and establish a correspondence to the noun that will fill this dependency. This is a case of the larger problem of *long range dependencies* in natural language. Relative sentences that act as noun modifiers is the only type of long range dependency we will explore in detail in this work.

Besides single pronouns, in Spanish it is possible to use more complex phrases to introduce a relative sentence, like in the following examples:

(34) *La ciudad en donde nací / The city where I was born*

(35) *La mujer cuyo auto compré / The woman whose car I bought*

Phrase 34 introduces the relative clause using the expression *"en donde"* (*"where"*), while phrase 35 uses the pronoun *"cuyo"* (*"whose"*) followed by a noun phrase. We will call single pronouns (like *"que"*) or other expressions (like *"en donde"* and *"cuyo auto"*) that can introduce a relative sentence **relative pronominal expressions**. The relative sentence introduced by one of these relative pronominal expressions will have a dependency with the context, and the expression will act both as an argument inside the relative sentence and as a modifier to a noun in the container sentence.

Relative pronominal expression

The `head_rel` rule, shown in figure 3.20, allows the combination of a relative pronominal expression with the relative sentence it introduces. Notice that this rule is very similar to the `head_mod` rule, because it is essentially an application of a modifier, with the difference that the non-local feature `REL` is used instead of the valence feature `MOD`.

$$(\mathrm{H})\ \boxed{1} + \begin{bmatrix} \mathrm{expr} \\ \mathrm{NONLOCAL} & \left[\mathrm{REL} \quad \left\langle \boxed{1} \right\rangle\right] \end{bmatrix} \rightarrow \boxed{1}$$

Figure 3.20: Feature structure for the relatives rule `head_rel`.

The way the `REL` feature works is different than the way the valence features work. After the application of any rule (except for `head_comp_sem` as seen in

section 3.5), the result of the rule will inherit all valence features from the head daughter that are not affected by the rule. This is how the standard *valence principle* works in HPSG. For the `REL` feature, however, the result will inherit the union of the `REL` values in the daughters, in the following way:

- If the `REL` feature is empty in both daughters, the `REL` of the result will be empty.
- If only one of the daughters has a non-empty `REL` feature, it is copied to the result.
- If both daughters have non-empty `REL` features, they must unify, and the unified expression is copied to the result.

The `REL` feature is generally empty for all lexical entries except relative pronouns (such as *"que"* and *"donde"*). Consider the phrase *"El libro que Juan leyó"* (*"The book that John bought"*) from sample 32, whose analysis is shown in figure 3.21. *"Juan leyó"* is analyzed as a verb phrase that is missing a complement. This complement is fulfilled by the relative pronoun *"que"* using the `comp_head` rule. This causes the `REL` feature that was in the pronoun to be percolated to the resulting verb phrase, which enables the application of the `head_rel` rule with the noun *"libro"*.

The application of the `head_rel` rule stops the percolation of the `REL` feature. The feature is no longer needed up the tree, as its constraints have already been satisfied. This works in a similar way to the `filler-gap` constructions in standard HPSG.

In the parse tree we can navigate from `ARG1` of the verb *"llegó"* to the relative pronoun *"que"*, and from there through the `REL` feature to the noun *"libro"*, so this analysis allows us to retrieve the correct argument for the verb in this non-local dependency construction.

Notice that in English it is possible to translate this sentence as *"The book John read"* omitting the relative pronoun. This is not possible in Spanish: relative sentences are always introduced by some kind of relative pronominal expression that will include a pronoun, so we can be sure the `REL` feature will be introduced by this pronoun.

The analysis of *"El perro que me mordió"* (*"The dog that bit me"*) from sample 33, as shown in figure 3.22, is essentially the same, but in this case the relative pronoun is acting as the subject of the relative sentence, so the `spec_head` rule is used.

The analysis of phrase 34 *"La ciudad en donde nací"* (*"The city where I was born"*), as shown in figure 3.23, presents some differences. First of all, instead of a single pronoun it uses the relative pronominal expression *"en donde"*. The pronoun *"donde"* is the one that introduces the `REL` feature, which is percolated to the prepositional phrase upon application of the `head_comp` rule.

The relative expression in this case is not acting as a subject or complement, but as a modifier of the relative sentence (it is not shown in the diagram but

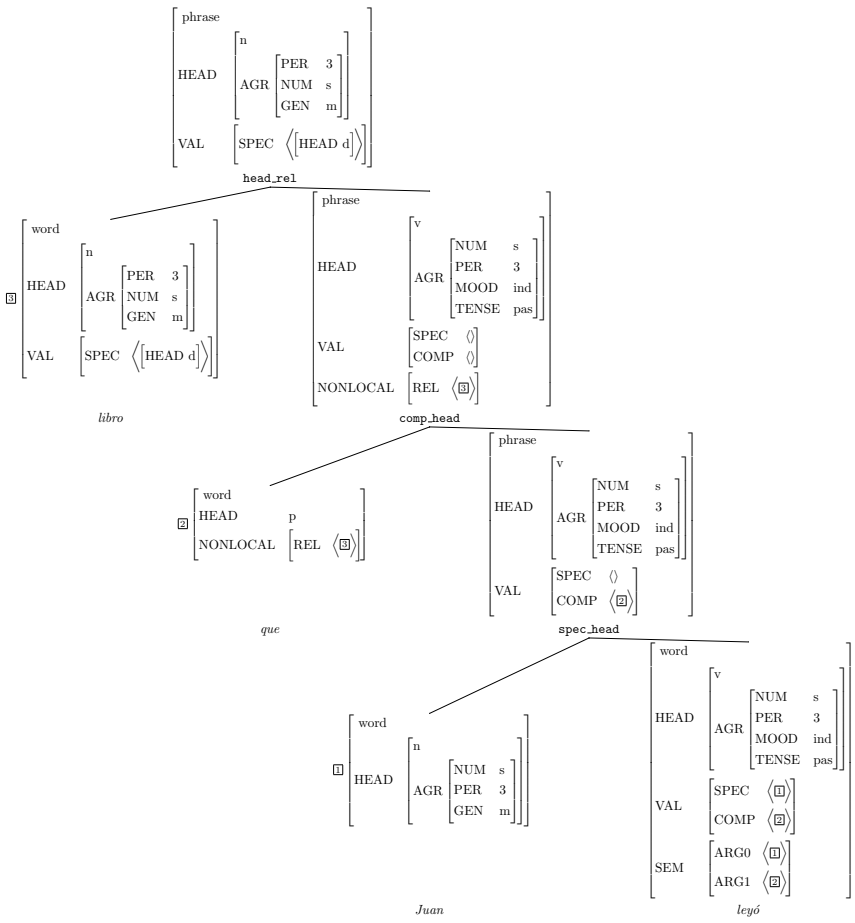


Figure 3.21: Analysis of a fragment of the noun phrase *"El libro que Juan leyó"* (*"The book that John read"*).

it becomes an `ARGM`). This is why we apply the rule `mod_head` to combine the expression to the relative sentence.

The analysis shown in figure 3.24 of phrase 35 *"La mujer cuyo auto compré"* (*"The woman whose car I bought"* has a complex relative pronominal expression that combines a pronoun with an argument. As usual, the pronoun introduces the `REL` feature, and this pronoun *"cuyo"* (*"whose"*) also expects a noun phrase as a complement. Once the complement is applied, the resulting phrase inherits the `REL` feature so it can be treated as any other relative pronominal expression: it is applied as a complement of the relative sentence, and the result is applied as a relative (modifier) of the noun.

We can summarize the life cycle of the `REL` feature during the parsing process as follows:

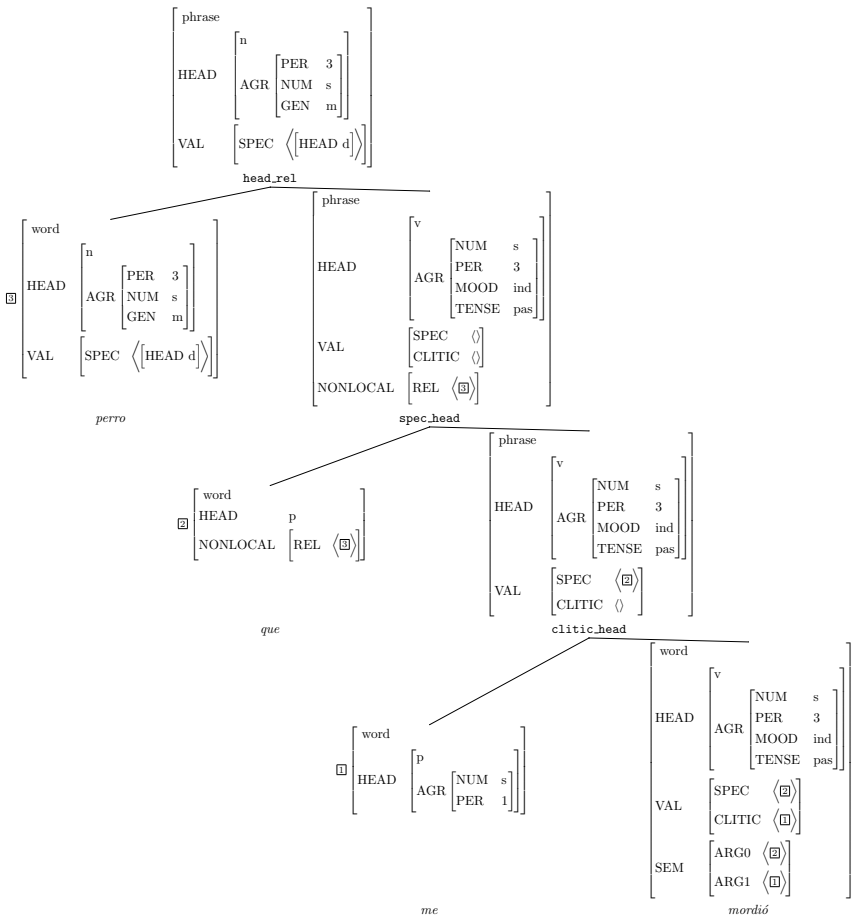


Figure 3.22: Analysis of a fragment of the noun phrase *"El perro que me mordió"* (*"The dog that bit me"*).

- The `REL` feature is always introduced by a relative pronoun.
- Any non-empty `REL` feature will be inherited by its parent in the tree. If both daughters of a node contain a non-empty `REL` feature, their values are unified.
- The relative pronominal expression will be attached to the left of a relative sentence, which implies applying one of the rules `spec_head`, `comp_head` or `mod_head`.
- The `head_rel` rule expects an expression with a non-empty `REL` feature on the right side, and its result clears the `REL` value so it is not inherited further up the tree.

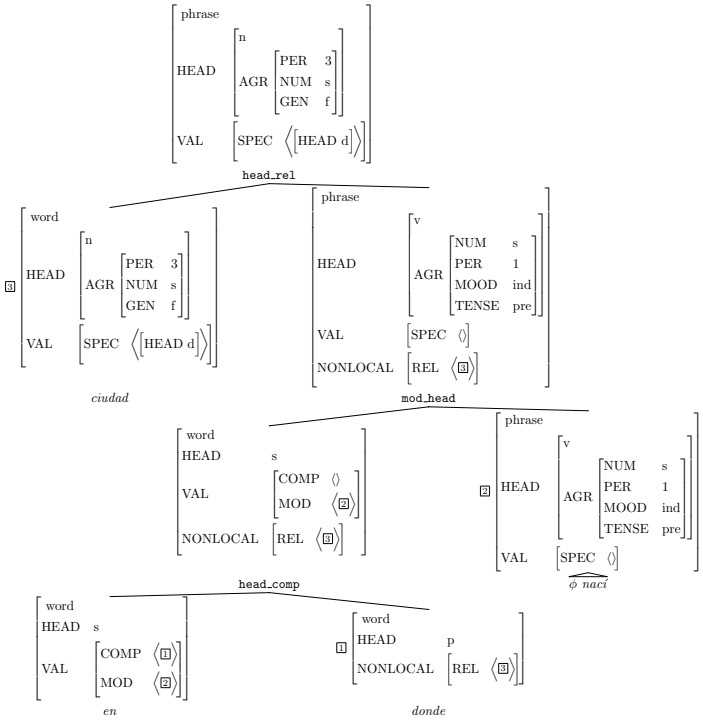


Figure 3.23: Analysis of a fragment of the noun phrase *"La ciudad en donde nací"* (*"The city where I was born"*).

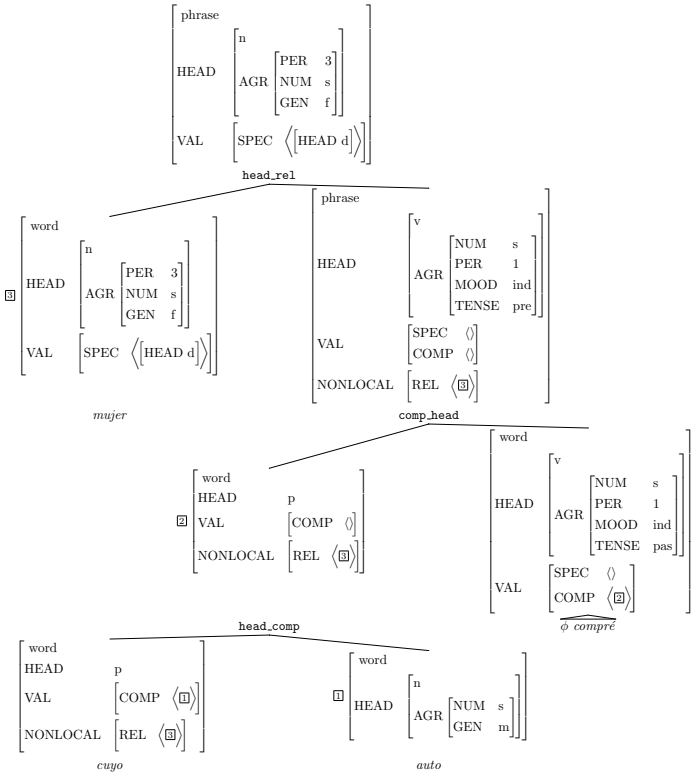


Figure 3.24: Analysis of a fragment of the noun phrase *"La mujer cuyo auto compré"* (*"The woman whose car I bought"*).

3.9 Punctuation rules

Any punctuation marker can be attached to any expression using one of the two head punctuation rules: `head_punct` and `punct_head`. Punctuation markers use the part of speech `f`, so any word marked as `f` can be used for this rule, no other feature is necessary.

However, certain punctuation markers can be used for building a coordination. For example, in the expression *"manzanas, naranjas y peras" ("apples, oranges and pears")*, the first "," is used as a coordination marker. These cases use the two coordination features, as explained in section 3.10, so we ask that for the standard punctuation rule, those features are defined as empty. This is shown in figure 3.25, which shows the definition of the two punctuation rules.

$$\begin{bmatrix} \text{word} & \\ \text{HEAD} & \text{f} \\ \text{VAL} & \begin{bmatrix} \text{LEFT\_COORD} & \langle\rangle \\ \text{RIGHT\_COORD} & \langle\rangle \end{bmatrix} \end{bmatrix} + \quad (\text{H})\ \boxed{1} \rightarrow \boxed{1}$$

(a) `punct_head`

$$(\text{H})\ \boxed{1} + \begin{bmatrix} \text{word} & \\ \text{HEAD} & \text{f} \\ \text{VAL} & \begin{bmatrix} \text{LEFT\_COORD} & \langle\rangle \\ \text{RIGHT\_COORD} & \langle\rangle \end{bmatrix} \end{bmatrix} \rightarrow \boxed{1}$$

(b) `head_punct`

Figure 3.25: The left-headed and right-headed versions of the punctuation rule.

3.10 Coordination rules

Coordinations are associations between two or more compatible structures that combine to form a structure with similar combinatorial properties as its parts. For example it is possible to coordinate two noun phrases to generate another noun phrase. In Spanish a coordinated structure always uses another grammatical element that acts as connecting tissue between the coordinated elements: typically a conjunction or a punctuation sign (see section 3.9). The simplest example of this is the coordination between two structures using a conjunction, such as the coordinated noun phrase *"manzanas y naranjas"* (*"apples and organes"*).

Coordination chain

It is possible to connect an unbounded number of compatible elements in a coordination, for example separating them by a comma: *"duraznos, peras, manzanas y naranjas"* (*"peaches, pears, apples and oranges"*). In this case we say there is a **chained coordination**, that could be easily decomposed into a series or applications of simpler coordinations: *"duraznos"* is coordinated to *"peras, manzanas y naranjas"*; which is *"peras"* coordinated to *"manzanas y naranjas"*; which is *"manzanas"* coordinated to *"naranjas"*.

Each of these simpler coordinations are indeed ternary structures that combine:

- A left coordinated element.
- A connector: typically a conjunction (like *"y"* / *"and"*) or another element that can act as connecting tissue for the coordination (such as a comma).
- A right coordinated element that is compatible with the left one. This element might be another coordinated structure, allowing the formation of chained coordinations.

As our grammar only contains binary rules, we need a way to binarize this kind of ternary structures. In order to do this, we define two rules that must be applied one after the other to form a coordination: `coord_left` and `coord_right`. `coord_right` attaches a conjunction (or another coordinating connector) to the right side of the coordination yielding a incomplete coordination, while `coord_left` attaches a left element to an incomplete coordination to make it complete. Figure 3.26 shows the two rules, and figure 3.27 shows the analysis of the *"manzanas and naranjas"* example.

The different elements in a coordination are said to be compatible, but they do not necessarily have the same POS. Consider the following example:

(36) *La cerca es blanca y de madera (The fence is white and wooden)*

Example 36 contains the expression *"blanca y de madera"* that coordinates an adjective and a prepositional phrase. Both elements are compatible in the sense that both define properties of the fence. In our grammar, the connector is the key element for modelling this behavior. Unlike standard HPSG, we include two new valence features in the connectors intended to specify the type

$$\begin{bmatrix}\text{expr}\\ \text{VAL}\left[\text{RIGHT\_COORD}\left\langle \boxed{1}\right\rangle\right]\end{bmatrix} + \text{(H)}\,\boxed{1} \rightarrow \begin{bmatrix}\text{phrase}\\ \text{VAL}\left[\text{RIGHT\_COORD}\,\langle\rangle\right]\end{bmatrix}$$

(a) `coord_right`

$$\text{(H)}\,\boxed{1} + \begin{bmatrix}\text{expr}\\ \text{VAL}\begin{bmatrix}\text{LEFT\_COORD}\left\langle \boxed{1}\right\rangle\\ \text{RIGHT\_COORD}\,\langle\rangle\end{bmatrix}\end{bmatrix} \rightarrow \begin{bmatrix}\text{phrase}\\ \text{VAL}\begin{bmatrix}\text{LEFT\_COORD}\,\langle\rangle\\ \text{RIGHT\_COORD}\,\langle\rangle\end{bmatrix}\end{bmatrix}$$

(b) `coord_left`

Figure 3.26: The left and right coordination rules.

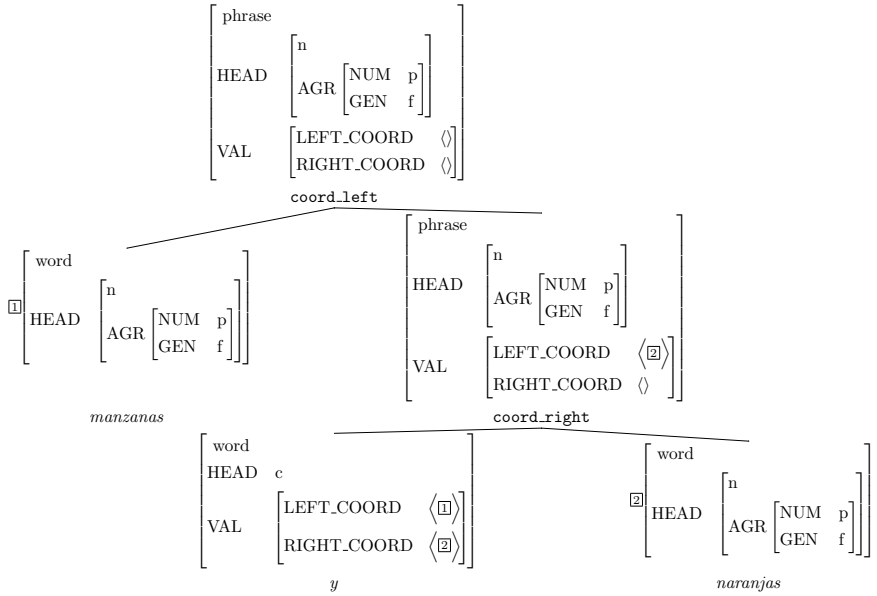


Figure 3.27: Analysis of the coordination *"manzanas y naranjas"* (*"apples and oranges"*).

of elements the connector is meant to coordinate: `LEFT_COORD` and `RIGHT_COORD`. This will of course depend on the context of the sentence. The lexical entry for the conjunction *"y"* in example 36 is shown in figure 3.28, and the resulting coordination is shown in figure 3.29.

As shown in figure 3.26 by convention, the head of the `coord_right` rule is the right expression, and the head of the `coord_left` rule is the left expression. This means that, after creating a chain of coordinations, the leftmost element of the chain will be its main head.

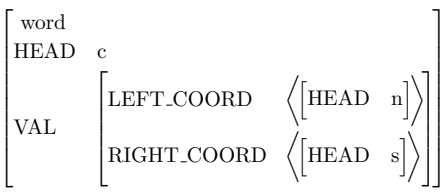


Figure 3.28: Feature structure for conjunction *"y"* in *"la cerca es blanca y de madera"*.

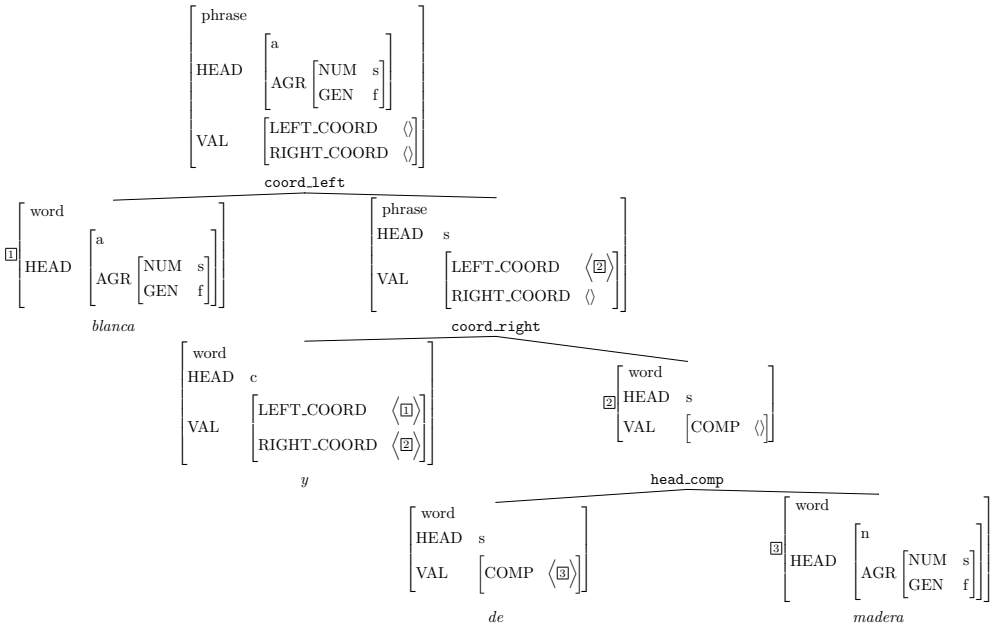


Figure 3.29: Analysis of the coordination *"blanca y de madera"* in *"la cerca es blanca y de madera"*.

3.11 Basic implementation of a rule

As mentioned in section 3.1, the rule notation we used so far is a simplification. Rules are also implemented as feature structures, and the relationships between the features of the head and its daughters are defined in terms of feature constraints. As we consider all rules to be binary and each one of them identifies only one of their children as its head, we can define the simplest feature structure for a rule as the one shown in figure 3.30. This feature structure defines a `RULE_HEAD` that will be the head expression, and a `RULE_DEP` that will be the dependent expression. The head and the dependent each will contain an expression, and the result of the application of the rule will be another expression that is the `CONTENT` of the rule itself.

$$\begin{bmatrix} \text{rule} & \\ \text{RULE\_HEAD} & \begin{bmatrix}\text{CONTENT} & \text{expr}\end{bmatrix} \\ \text{RULE\_DEP} & \begin{bmatrix}\text{CONTENT} & \text{expr}\end{bmatrix} \\ \text{CONTENT} & \text{expr} \\ \text{RULE\_NAME} & \text{text} \end{bmatrix}$$

Figure 3.30: Basic feature structure for an abstract rule.

As we have seen, most of the rules have left-headed and right-headed variants. It is easy to implement this by having two features `LEFT` and `RIGHT` in the rule and coindexing them with the `RULE_HEAD` or `RULE_DEP` features accordingly, as shown in figure 3.31.

$$\begin{bmatrix} \text{rule} & \\ \text{RULE\_HEAD} & \boxed{1}\begin{bmatrix}\text{CONTENT} & \text{expr}\end{bmatrix} \\ \text{RULE\_DEP} & \boxed{2}\begin{bmatrix}\text{CONTENT} & \text{expr}\end{bmatrix} \\ \text{LEFT} & \boxed{1} \\ \text{RIGHT} & \boxed{2} \\ \text{CONTENT} & \text{expr} \\ \text{RULE\_NAME} & \text{text} \end{bmatrix}$$

(a) Left-headed rule.

$$\begin{bmatrix} \text{rule} & \\ \text{RULE\_HEAD} & \boxed{1}\begin{bmatrix}\text{CONTENT} & \text{expr}\end{bmatrix} \\ \text{RULE\_DEP} & \boxed{2}\begin{bmatrix}\text{CONTENT} & \text{expr}\end{bmatrix} \\ \text{LEFT} & \boxed{2} \\ \text{RIGHT} & \boxed{1} \\ \text{CONTENT} & \text{expr} \\ \text{RULE\_NAME} & \text{text} \end{bmatrix}$$

(b) Right-headed rule.

Figure 3.31: Feature structure for the abstract left-headed and right-headed rules.

Notice how the feature structures in figure 3.31 match the simplified notation shown in figure 3.1: The daughters (on the left of the arrow in figure 3.1) are the expressions marked as `LEFT` or `RIGHT` in the feature structure. The head marker '(H)' is represented by coindexing the features `RULE_HEAD` and `RULE_DEP` accordingly. Finally, the resulting expression on the right side of the arrow in figure 3.1 is the expression pointed by the root feature `CONTENT` in figure 3.31.

3.12 Principles

HPSG encodes some of its constraints as principles Sag et al. [2003]. Some of these principles can be implemented directly into the feature structures and some are used as a guideline for implementing the features or rules. In our grammar we use the following principles:

Head Feature Principle

The `HEAD` feature of the daughter marked as head should be the same as the `HEAD` daughter of the resulting phrase. This is the same as the Head Feature Principle used in [Sag et al., 2003]. It can be easily implemented by unifying all rules with the structure shown in figure 3.32, which ensures that the `HEAD` feature of the daughter is unified with the `HEAD` feature of the resulting phrase.

$$\begin{bmatrix} \text{RULE\_HEAD} & \begin{bmatrix}\text{CONTENT} & \begin{bmatrix}\text{HEAD } \boxed{1}\end{bmatrix}\end{bmatrix} \\ \text{CONTENT} & \begin{bmatrix}\text{HEAD } \boxed{1}\end{bmatrix} \end{bmatrix}$$

Figure 3.32: Implementation of the Head Feature Principle.

Valence Principle

Unless otherwise stated by the rules, the features inside the `VAL` feature from the daughter should be the same for the resulting phrase. This is analogous to the Valence Principle used in [Sag et al., 2003]. Notice, however, that this principle indicates a default value, but the different rules should change this behaviour depending on their needs. Because of this, it is not possible to implement it as a single feature structure.

In our case, we copy the values of the `SPEC`, `COMP`, `CLITIC`, `MOD`, `LEFT_COORD` and `RIGHT_COORD` from the head daughter to the resulting phrase, except for the features explicitly modified by the rule. For example, the specifier rules should copy all the rest of the `VAL` features but modify only the `SPEC` value in the phrase.

An important exception to this principle is the `head_comp_sem` rule, which is designed explicitly to copy the valence features from the dependent instead of the head daughter.

Agreement Principle

As mentioned in section 2.1.4, HPSG defines an agreement principle that allows us to model agreement constraints without the complexities that would happen in CFG. [Sag et al., 2003] defines the Specifier-Head Agreement Constraint, which states that if a lexical element requires a specifier, it should agree with its specifier. This principle was formulated for English, but it is true in Spanish

as well. Also, for Spanish we would like an adjective to agree with the head noun in a noun phrase construction. As adjectives are attached using the `MOD` feature, we would also need to indicate that the agreement principle should be honored when applying a modifier.

In terms of implementation as feature structures, our version of the agreement principle implies unifying the specifier and modifier rules with the structure shown in figure 3.33, which ensures that for those rules the `AGR` feature in the dependent unifies to the one in the head.

$$\begin{bmatrix} \text{RULE\_HEAD} & \begin{bmatrix} \text{CONTENT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{RULE\_DEP} & \begin{bmatrix} \text{CONTENT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 3.33: Implementation of the Agreement Principle.

As both number and gender are encoded as features in `AGR`, this principle effectively works for modeling number and gender agreement in Spanish. However, see section 4.7 for a discussion of the behavior of this principle in an actual corpus.

Relative Principle

As seen in section 3.8, the `REL` feature works differently than the valence features. We want that if the `REL` feature has a non-empty value in either the head or the dependent, this value is percolated to the resulting phrase, otherwise the empty `REL` is kept.

This is similar, but not exactly the same, as the Gap Principle in [Sag et al., 2003], which states that the `GAP` feature of the daughters are added up in the `GAP` feature of the parent. The difference is that we are not allowing many `REL` expressions, as would happen with gaps, so if two expressions expect a `REL`, both values would be unified. In practice this works well because the `REL` feature is generally fulfilled by expressions that are relatively close in the tree.

This principle is not implemented as a single feature structure, but is enforced during the rule applications.

Chapter 4

Description of the Corpus

This chapter describes the way we created a corpus of sentences analyzed according to our HPSG grammar. The corpus is based on the AnCora corpus (see section 2.4.2) and was transformed using a set of heuristics and later analyzed using a feature structure implementation of our grammar. We also provide some statistics of interest for the corpus.

4.1 Initial corpus transformation

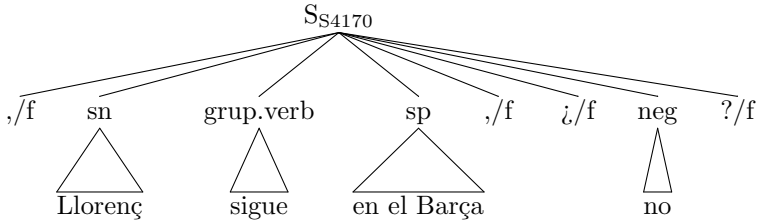
Elementary HPSG tree

The initial transformation of the Spanish AnCora corpus is described in [Chiruzzo and Wonsever, 2016; Chiruzzo, 2015]. The transformation process comprises a top-down step followed by a bottom-up step, with the aim of transforming all sentences in AnCora into **elementary HPSG trees**, i.e. a head surrounded by its direct dependents (complements, specifier, modifiers).

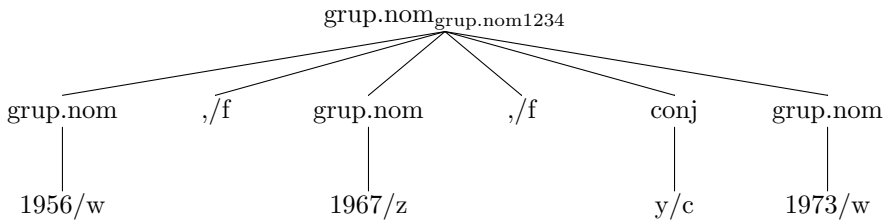
The top-down step is in charge of simplifying some complex structures found in the corpus. As mentioned in section 2.4.2, AnCora contains many variants of rules for some of its categories like `S` (subordinate sentence), `sn` (noun phrase) or `grup.nom` (nominal group). Much of this variability can be explained because the tree structures in AnCora tend to be rather flat. Many structures like nested blocks, subclauses, coordinations and some punctuation lack a uniform annotation throughout the corpus, and are often attached to the same parent without deeper analysis. Figure 4.1 shows some examples of AnCora nodes with several children. In the first case, 4.1a, we see a subordinate sentence *", Llorenç sigue en el Barça, ¿no?"*, which contains as children the subject, main verb, the prepositional complement, a modifying block, and several punctuation symbols. As can be seen in the example, the punctuation symbols and some constructions do not follow a strict annotation logic: the first comma (separating from a previous sentence) is located inside the subordinate sentence, but the question marks associated to the nested block *"¿no?"* are outside the block.

The second example, 4.1b, is a coordination of three elements. When modeling coordinations in AnCora the usual structure is a parent category that con-

tains any number of subtrees for the coordinated elements and their connectors in a flat arrangement. The figure shows the original structure of `grup.nom1234` *"1956, 1967, y 1973"* in the corpus. We can see that the direct children of the top `grup.nom` node are a flat mixture of other `grup.nom`s, punctuation symbols and a conjunction. There is a rule variant for this structure that coordinates three groups with this combination of separators, and also variants for chains of coordinations with many numbers of groups and different separators.



(a) Subordinate sentence `S4170`.



(b) Nominal group `grup.nom1234`.

Figure 4.1: Examples of original structures found in AnCora prior to the transformation.

For modeling these sentences in our grammar we need them to be strictly binary, and this kind of complexities represent challenges in the transformation process. The top-down process tries to break down these complex structures and leaves simpler units that can be tackled by the bottom-up process. The bottom-up process assumes that the structures left behind represent flattened elementary HPSG trees, and is in charge of finishing the binarization and detection of arguments. This process uses a set of manually written heuristic rules for deciding which the elements inside the phrase is the head, what rules should be applied to relate this head to the rest of the elements, and in which order they should be applied so the constituent is correctly binarized. The rules are written in a simple language, designed for that purpose, for defining the categories and possible attributes that could be used to identify heads or arguments. For example, table 4.1 shows the head finding rules used for the `grup.nom` category. The rules are applied in order, so the first one has higher priority than the last one. In this example it means that finding a noun (`n`) as head has precedence over finding a number (`z`), an adjective (`a`) or a participle (`participi`). This initial transformation process achieved an average 95.3% precision for head de-

tection (98.7% without considering coordinations) and 92.5% average precision for argument detection [Chiruzzo and Wonsever, 2016].

- `n` (sustantivo)
  "... Río_Bravo y Saltillo para la [[$_H$ compañía] [francesa]]..."
- `grup.nom` (grupo nominal anidado)
  "... y sobre [[$_H$ transmisiones y retenciones] [de fondos de inversión]] ."
- `p` (pronombre)
  "... obtuvo 19 diputados, [[$_H$ dos] [más]] que en 1996..."
- `w` (fecha)
  "... hundimiento del "Kursk" el [[pasado] [$_H$ 12_de_agosto]] en aguas árticas..."
- `z` (número)
  "... donde lograron el [[$_H$ 71_por_ciento] [de los sufragios]] ..."
- `a` (adjetivo)
  "... quien cuestiona al entrenador es [[$_H$ enemigo] [del Barça]] ."
- `v` (verbo)
  "... sobre todo en el [[$_H$ capitulo] [de las infraestructuras]] ..."
- `s.a` (sintagma adjetival)
  "... y la [[$_H$ segunda] [, mucho más potente,]] a las 07.30.42..."
- `participi` (participio)
  "... el relato ZZadjNM de lo [[$_H$ ocurrido] [en la sima de ZZlugar]] ..."
- `S/clausetype=participle` (oración subordinada de tipo participio)
  "... en_lugar_del [[$_H$ destituido] [Carlos_Sainz_de_Aja]] ."
- `S/clausetype=relative` (oración subordinada de tipo relativa)
  "... incluidos los [[$_H$ que él mismo ha hablado] [sobre sí mismo]] ..."
- `S/clausetype=completive` (oración subordinada de tipo completiva)
  "Al [[$_H$ correr] [de los siglos]] se había manifestado un..."
- `sp` (sintagma preposicional)
  "aeropuerto de Miami, uno de los [[$_H$ de mayor tráfico aéreo] [en EEUU]]..."
- `sn` (sintagma nominal)
  "... el hotel (un [[$_H$ cinco estrellas de gran lujo]])..."

Table 4.1: Head finding rules for the `grup.nom` category.

4.2 Verb phrases

Special care had to be taken for the transformation of verb phrases, which include auxiliary and modal verb constructions, because their analysis in AnCora was different than the rest of the phrases. The `grup.verb` category in AnCora includes single verbs and also verbal periphrases like *"pueden hacer"* (*"can make"*) or *"han comenzado a cantar"* (*"have begun to sing"*). Keeping this behavior found in the AnCora corpus is the reason why decided to model verbal periphrases as units, as described in section 3.5. If we consider a sentence like *"ellos pueden hacer pasta"* (*"they can make pasta"*), a standard HPSG analysis for this phrase would first apply the complement *"pasta"* to the

verb *"hacer"*, then this verb would be applied as a complement to the verb *"pueden"* and finally the subject *"ellos"* would be applied to the resulting head. However, constructions of this type are analyzed differently in AnCora: the verbal periphrasis is considered a unit, so *"pueden hacer"* becomes a phrase that should expect a complement and a subject. Considering verbal periphrases as units simplifies the analysis of displaced constituents, so we kept this behavior in our grammar and designed the `head_comp_sem` rule that percolates the verbal arguments from the dependent to the parent of the rule in order to model it homogeneously in our framework. Figure 4.2 shows the analysis of the verbal periphrasis *"pueden hacer"* as it would be used in the sentence *"ellos pueden hacer pasta"*.

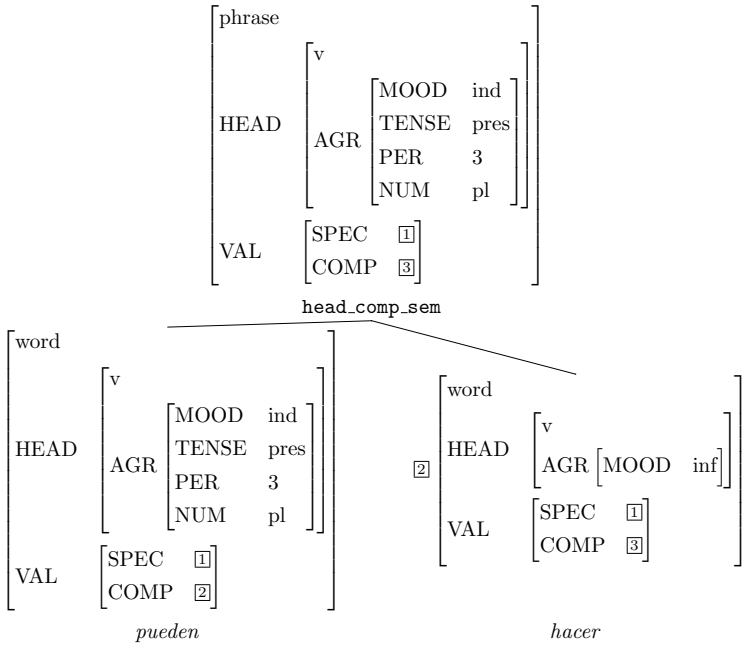


Figure 4.2: Analysis for verb phrase *"pueden hacer"* (*"can make"*).

The transformation of verb phrases was forcefully different than the transformation of other categories, as a `grup.verb` does not correspond to the "head surrounded by arguments" paradigm. We wrote a set of fifteen templates for identifying different types of `grup.verb` instances, with a particular way of transformation for each. The result is always a structure that has the leftmost verb as syntactic head (the support verb), but expects the arguments of the rightmost verb (the content verb).

4.3 Clitics

Clitics also needed a special treatment, as described in [Chiruzzo and Wonsever, 2018]. In order to model the Spanish clitic system we use the `clitic_head` rule, which allows us to attach a verb head to a clitic pronoun on its left, as mentioned in section 3.7. This rule is used for modeling any clitic pronoun that is attached directly to the verb, whether it is used for marking a pronominal verb, or as a verb argument. The difference between them is that the verb arguments will also be coindexed with a semantic argument inside the verb. Furthermore, if the sentence contains clitic reduplication, the `ARG` feature will have two values in its list: the clitic and the explicit argument. For example in the sentence *"Juan le dará un regalo a María"* (*"John will give a present to Mary"*), the clitic (*"le"*) corresponds to the indirect object (*"a María"*) which is also present in the sentence. The lexical entry for the verb *"dará"* (*"will give"*) as used in this sentence is shown in figure 4.3. Notice that both the clitic and the prepositional complement are set as `ARG2` in the argument structure, which corresponds to the beneficiary semantic role. If either the clitic or the explicit argument are present, then the semantic argument will point to that expression, if both are present then the list associated to the semantic argument will contain both expressions. AnCora provides a semantic role attribute for clitics and for other expressions that can be used to identify the appropriate semantic feature.

$$\begin{bmatrix} \text{word} & \\ \text{HEAD} & \begin{bmatrix} \text{v} & \\ \text{AGR} & \begin{bmatrix} \text{MOOD} & \text{ind} \\ \text{TENSE} & \text{fut} \\ \text{PER} & 3 \\ \text{NUM} & \text{sg} \end{bmatrix} \end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \text{SPEC} \left\langle \boxed{1} \left[\text{HEAD n}\right] \right\rangle \\ \text{COMP} \left\langle \begin{matrix} \boxed{2}\left[\text{HEAD n}\right], \\ \boxed{3}\left[\text{HEAD s}\right] \end{matrix} \right\rangle \\ \text{CLITIC} \left\langle \boxed{4} \right\rangle \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{ARGS} & \begin{bmatrix} \text{ARG0} & \left\langle \boxed{1} \right\rangle \\ \text{ARG1} & \left\langle \boxed{2} \right\rangle \\ \text{ARG2} & \left\langle \boxed{3}, \boxed{4} \right\rangle \end{bmatrix} \end{bmatrix} \\ \text{TEXT} & \textit{dará} \end{bmatrix}$$

Figure 4.3: Feature structure for ditransitive verb *"dará"*, future indicative third person singular form of the verb *"to give"*

4.4 Relatives

As mentioned in section 3.8, the only long range dependencies we focus on in this work are the relative clauses used as modifiers of nouns. In these cases, the noun is at the same time acting as an argument of the verb in the clause besides being modified by it. Generally the noun acts as the subject (e.g. *"el perro que me mordió"* / *"the dog that bit me"*) or direct object (e.g. *"el libro que leí"* / *"the book I read"*), but it could act as any argument. Unlike English, in Spanish it is mandatory that these clauses are introduced by a subordinating relative expression that always contains a relative pronoun, such as *"que"* (*"that"*) or *"a quien"* (*"to whom"*), which makes these expressions the ideal hooks for including the `REL` feature that models this behavior.

AnCora also provides some information for other kinds of long range dependencies like discontinuities. However, the analysis of these constructions entails further complexities that should be explored in detail, so the modeling of these in our grammar was left out of the scope of this work.

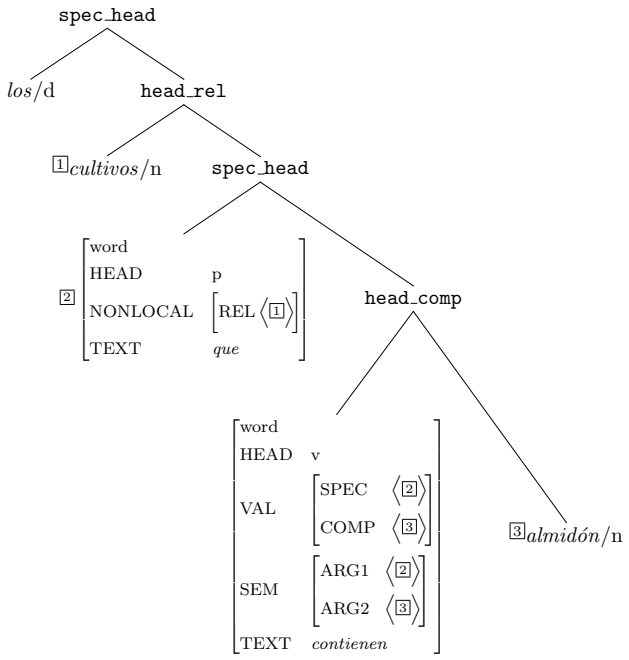


Figure 4.4: Simplified analysis for *"los cultivos que contienen almidón"* (*"the crops that contain starch"*).

Figure 4.4 shows the analysis according to our grammar for the example noun phrase with id `sn29674` originally in AnCora: *"los cultivos que contienen almidón"* (*"the crops that contain starch"*). The verb of the relative clause is transitive, but its corresponding subject (*"cultivos"*) is not readily available. Instead, the relative pronoun *"que"* takes the place of the subject, but keeps a

non-local feature REL that points to the noun it stands for. The rule head_rel is used to unify a non-saturated NONLOCAL.REL feature to the appropriate expression it should be bound to, the resulting phrase cancels the value of the NONLOCAL.REL feature. The semantic role information for these structures is also present in AnCora, encoded as an attribute in the relative expression. These attributes were leveraged during the transformation process and identified as semantic arguments of the corresponding verb inside the relative clause.

4.5 Null subjects

As seen in section 2.4.2, AnCora marks null subjects in a sentence as a sn structure with a special attribute. We kept this behavior in our corpus, transforming all instances of null subjects to a special empty lexical entry (its text value is *"()"*, as this sequence of tokens is not used in the AnCora corpus). This allows us to model null subjects in the same way as AnCora does.

However, we must take into account that a parser trained using these marks will have lower performance when presented with an input text that does not have them. One way of solving this issue is preprocessing the inputs of the parser and artificially adding the null subject marks. Doing this accurately is not easy, and it is an interesting area of research (see [Rello et al., 2012; González and Martínez, 2018]), but we left this treatment out of the scope of the current work.

4.6 Analysis with HPSG grammar

The result of the transformation process is a set of sentences containing information on each lexical entry, the values for some features, and the rules to apply to them so as to build each sentence as an HPSG tree. We then processed each one of the trees restoring the corresponding valence features for each lexical entry. This process is relatively simple: Based on the rules applied to each of the structures, we recursively transverse from the root of the sentence to the leaves inferring the features that the lexical entries should have. The process collects the rules applied in a chain of nested heads, and starts over for each dependent (non-head element). We infer the valence features from this sequence of rules applied from the root to a leaf. Special care has to be taken for dealing with the following constructions: for coordinations, the features should be replicated on the different branches; and when handling relative constructions, we must ensure that the relative pronouns end up with the REL feature, but not the rest of the words.

In order to validate our HPSG grammar and evaluate the factibility of using it on this corpus, we implemented the grammar using the feature structure implementation provided by the nltk library[1]. This feature structure library contains the basic principles for describing feature structures with unification

[1] http://www.nltk.org/howto/featstruct.html

and coindexation, but it is not designed to specifically enforce structure typing, unlike the one used in standard HPSG theory. This means we are using feature structures without a type hierarchy, but in our case this is not a problem because we use the corpus to generate the lexical entries. The type hierarchy becomes a key feature when developing a manual grammar because it gives the developer the flexibility to encode the linguistic information elegantly in types and apply them throughout the grammar. However, in our case we extract the grammar information directly from the corpus, for example the lexical entries are all found in the corpus and they can be modeled with the exact level of detail that the corpus provides.

Table 4.2 shows the number of lexical frames (superclass of lexical entries without considering the word and the morphological features), lexical entries and instances discriminated by part of speech.

POS	Frames	Entries	Instances
a - adjective	120	11,697	35,939
c - conjunction	191	796	27,067
d - determiner	11	303	76,135
f - punctuation	57	136	65,546
i - interjection	8	62	99
n - noun	226	34,193	121,113
p - pronoun	38	431	22,692
r - adverb	74	2,214	18,952
s - preposition	227	1,841	79,901
v - verb	2,234	28,486	61,699
w - date	21	1,043	2,731
z - number	38	2,392	5,363
Total	3,245	83,594	517,237

Table 4.2: Number of lexical frames, entries and instances for each part of speech.

The feature structure library allowed us to implement both the lexical entries and the rules as feature structures. We used these to create a feature structure version of all the trees in the corpus, by building each lexical entry and applying the corresponding rules for building the tree. The aim of this exercise was to validate the grammar and create a final version of the corpus annotated in a feature structure format.

During this process, we detected and corrected errors in the annotations of words and constituents that impeded the proper application of the rules. Only 232 errors (e.g. in lexical entries or rule applications) had to be manually corrected, which is less than 0.05% of the total number of tokens. However, there was one structure in particular that yielded a great number of errors, as we will see in the following section.

4.7 Analysis of the agreement principle

We noticed a great number of analysis errors that occurred due to the agreement principle as described in section 3.12, i.e. checking that the specifier (or subject) agrees with the head in morphological features like person, number or gender (in Spanish we would also require that an adjective agrees with a modified noun). We ran the analyzer for a sample of 1,000 sentences and found that roughly 10% of those sentences failed due to agreement errors. When manually analyzing those errors, we found out there were several situations in which the agreement principle fails for our corpus using our grammar. Some of these situations are due to the nature of the corpus, while others happened because of the grammar design. Let us see examples of some of the situations in which the agreement principle in the way we implemented it fails:

- Subordinate sentences

  [$_H$ no será una blasfemia] [$_S$ llevar adelante la operación de rescate]
  wouldn't it be blashpemy to perform this rescue operation

  The subject in this sentence is a subordinate sentence. In these cases trying to enforce agreement between these structures would not make sense. The `AGR` feature in the subject will generally carry information that contradicts the `AGR` feature in the head, for example the verb tense or time.

- Coordinations

  In a coordination chain, we take as representative the leftmost element in the chain, and we are not doing any particular treatment for changing their agreement features of the chain (for example from singular to plural). This means that, on occasions, the coordination will not agree with the original context. This could happen when a coordinated chain like is used as subject, for example *"Juan y María corren"* (*"John and Mary run"*), if we take only *"Juan"* (singular) as representative of the coordination, it will fail the agreement with *"corren"* (plural).

- Noun phrases with units

  [$_S$ un centenar de artistas] [$_H$ fueron] / a hundred artists went
  [$_S$ más de un millón de espectadores] [$_H$ lo vieron] / more than a million spectators saw it
  [$_S$ la mayoría de los presentes] [$_H$ manifestaron su apoyo] / most of those present expressed their support
  [$_S$ el resto de imputados] [$_H$ se han acodigo a su derecho] / the rest of the accused exercised their right

  In these cases the head word is the unit (*"centenar"*, *"millón"*, *"mayoría"*, *"resto"*) which does not agree with the plural verbs.

- Annotation errors in AnCora

  On several occasions the sentence was correct but there were some errors in the features assigned to the words originally in AnCora, for example:

  [S las] [H generales de hace cuatro años] / the general (elections) four years ago
  The word *"generales"* is marked as male, though it is an adjective talking about the elections (female).

  [S las] [H navieras de japón] / the shipping companies from Japan
  The word *"navieras"* is marked as singular.

- Errors in the original text

  [S la] [H bolsas españolas] / the Spanish stock exchanges
  [S la siniestralidad de las carreteras argentinas] [H elevan a 10.000 el número de víctimas] / the accident rate of Argentine roads raises the number of victims to 10,000

  The original text violates the agreement principle, but ideally the system should be able to recover from these errors and provide a suitable analysis.

- Use of possessive pronouns

  Possessive pronouns have the person agreement feature (*"mi"* is 1st person, *"tu"* is 2nd person, etc.) provided by the morphological information. Of course nothing indicates that this person feature should agree with the person feature of the verb they will be transitively bound to. Using this feature values out of their context is an error.

- Gender of determiners

  [S el] [H agua] / the water
  [S el] [H alza más importante] / the most important rise
  [S los] [H cabezas de listas] / the heads of lists

  In Spanish there are rules for changing the gender of the determiner on some occasions for tonal reasons, these rules are very hard to model in this context.

- Relatives and ellipses

  [S la] [H que tiene el ave] / the one the bird has
  [S las] [H que rige este mercado] / the ones governed by this market

  These are rare cases in which the rule application would unify the male feature of *"bird"* or *"market"* with the verb (because they are the corresponding subjects) and then try to unify the relative sentence with the determiners (which are female) and fail.

These are the most frequent situations, although there are a few more. Some of them occur due to use of language, others because of flaws in our grammar, and others are real errors in the data that a parser should try to recover from.

We could try to tune the rules further in order to model some of these situations better, which would imply making the rules more complex. In this work, however, we decided to leave the more complex analysis as future work, and just disable the agreement principle, so the rules can accept all the faulty cases. It is clear that the agreement between subject and verb or between determiner and noun gives important information that a parser could use to identify the rules to apply, but given there are also plenty of exceptions, we expect that the statistical process for training the parser can learn to tell the different cases in which the agreement principle should or should not be honored directly from the data.

4.8 Statistics

The final corpus has approximately half a million words (517,237) in over 17,000 sentences. We split the contents of the corpus in training, development and test partitions according to the standard AnCora partitions used in CoNLL-2009 shared task [Hajič et al., 2009]: around 80% words for training, 10% for development and 10% for test. Table 4.3 shows the number of documents, sentences and words in each partition.

	Train	Dev	Test	Total
Documents	1,312	155	168	1,635
Sentences	14,018	1,638	1,692	17,348
Tokens	418,319	49,338	49,580	517,237

Table 4.3: Statistics of the corpus partitions in terms of number of documents, sentences and tokens.

POS	Train	Dev	Test	Total
a - adjective	29,011	3,494	3,434	35,939
c - conjunction	21,942	2,526	2,600	27,068
d - determiner	61,581	7,224	7,330	76,135
f - punctuation	52,919	6,296	6,330	65,545
i - interjection	85	1	13	99
n - noun	89,324	10,569	10,444	110,337
p - pronoun	18,418	2,122	2,152	22,692
r - adverb	15,360	1,776	1,816	18,952
s - preposition	73,252	8,780	8,645	90,677
v - verb	50,012	5,751	5,936	61,699
w - date	2,169	282	280	2,731
z - number	4,246	517	600	5,363
Total	418,319	49,338	49,580	517,237

Table 4.4: Number of word instances by part of speech for each corpus partition.

The number of words for each part of speech is shown in table 4.4, while the

number of times each rule is applied is shown in table 4.5.

Rule	Train	Dev	Test	Total
clitic_head	6,554	789	727	8,070
comp_head	4,679	544	553	5,776
coord_left	14,725	1,725	1,785	18,235
coord_right	14,725	1,725	1,785	18,235
head_comp	116,057	13,597	13,568	143,222
head_comp_sem	8,325	935	1,002	10,262
head_mod	74,171	8,979	8,825	91,975
head_punct	31,266	3,767	3,684	38,717
head_rel	6,269	731	737	7,737
head_spec	5,272	599	606	6,477
mod_head	22,520	2,662	2,691	27,873
punct_head	18,424	2,192	2,237	22,853
spec_head	89,964	10,501	10,742	111,207

Table 4.5: Number of times each rule is applied in the corpus for each corpus partition.

Figure 4.5 shows a histogram of sentence lengths for the whole corpus. The longest sentence in the corpus is 149 words long. We dropped sentences with only one word because they are trivial for parser development, so the shortest sentence is 2 words long. The majority of sentences (almost 80%) in the corpus are between 11 and 50 words long.

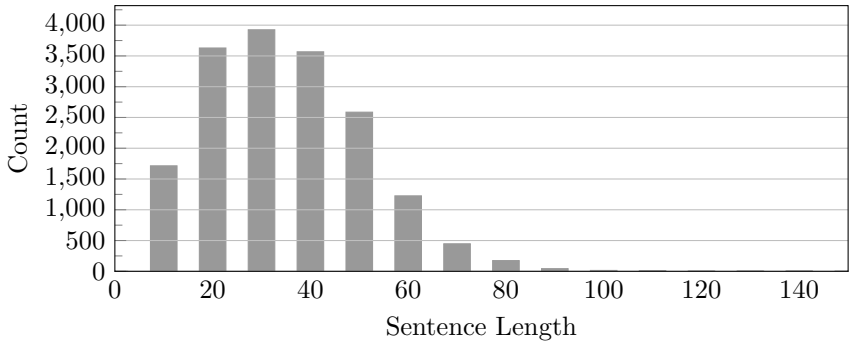


Figure 4.5: Distribution of sentences by length in the corpus.

Chapter 5

Parser development

In this chapter we present the parsers we implemented. We begin with a definition of the parsing problem, we outline some ways in which it could be solved, and discuss how to evaluate the performance of the parsers built for our grammar. Then we describe the different parsing strategies we tried for building our parser: some bottom-up parsing baselines of increasing complexity, a CKY approach, and a top-down approach. All the results presented in this chapter are evaluated against the development corpus, which we used for parameter tuning.

5.1 Definition of the problem

The aim of our parser is to transform a sentence into a tree encoded in our HPSG grammar. Given a sentence $S = \{w_1, w_2, \dots w_n\}$ find the HPSG tree $T \in \mathcal{T}$ that best represents S. $\mathcal{T}$ represents the set of possible trees that use the HPSG grammar defined in chapter 3.

This means taking as input a sentence like 37 and transforming it into the tree shown in figure 5.1.

(37) *El perro que vi estaba corriendo (The dog I saw was running)*

There are at least two ways to do this: we can start from the lexical entries and find the best tree, or we can try to find the best tree and infer the lexical entries from it. Both approaches have different advantages and disadvantages.

5.1.1 From lexical entries to trees

On the one hand, if we had the correct lexical entries for each one of the words in the sentence (the ones shown in figure 5.2), we could use an algorithm like CKY to find the correct rules to apply in order to get the correct tree.

For this approach, we need a process for assigning the correct lexical entries for each word. A grammar like ours generally has many possible lexical entries for each word, and some particularly ambiguous words could have hundreds of

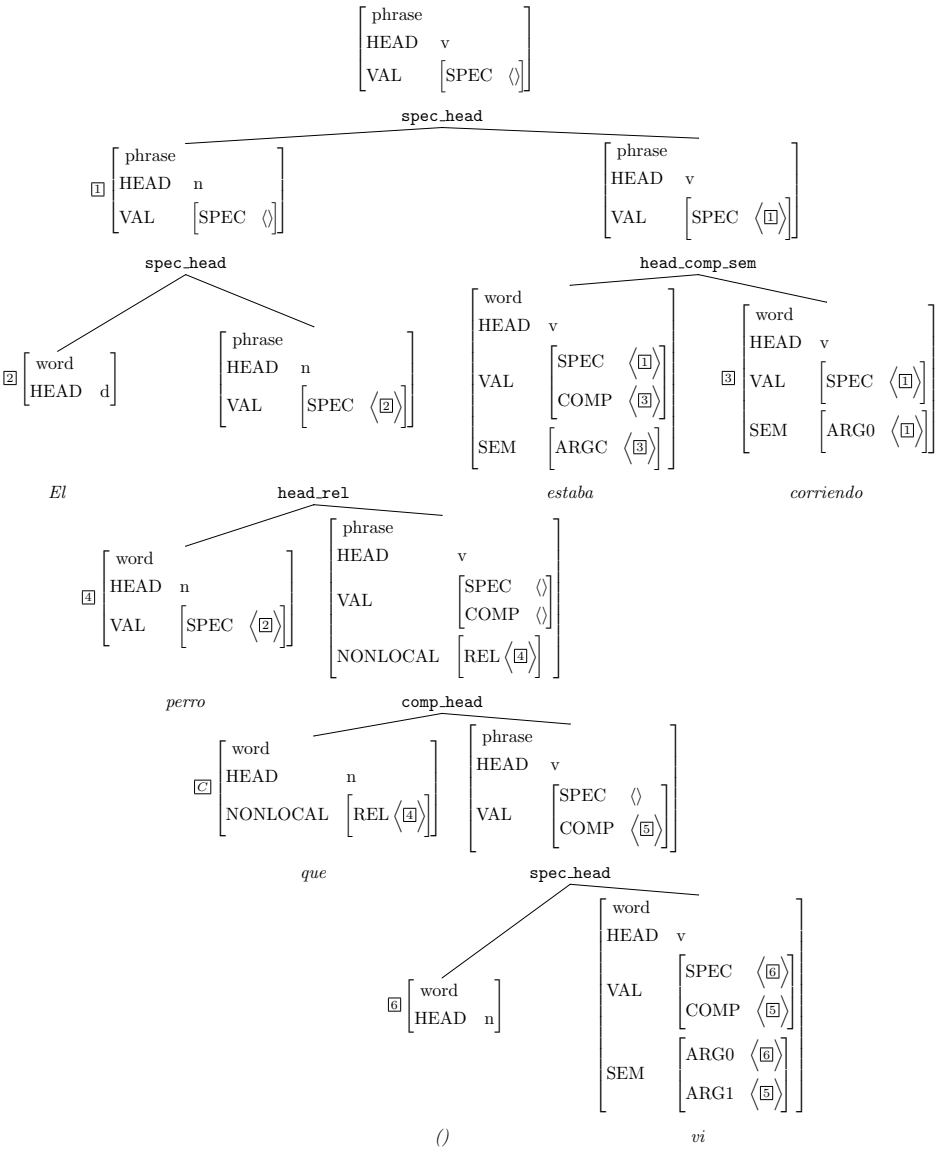


Figure 5.1: Expected parse tree for the sentence *"El perro que vi estaba corriendo"* (*"The dog I saw was running"*).

possible lexical entries. The use of a supertagger becomes essential if we want this approach to be tractable.

The main drawback of this is that a supertagger in general does not guarantee that the sequence of tags obtained will be compatible to form a valid tree, let alone the correct tree. It is possible that the supertagger returns a sequence

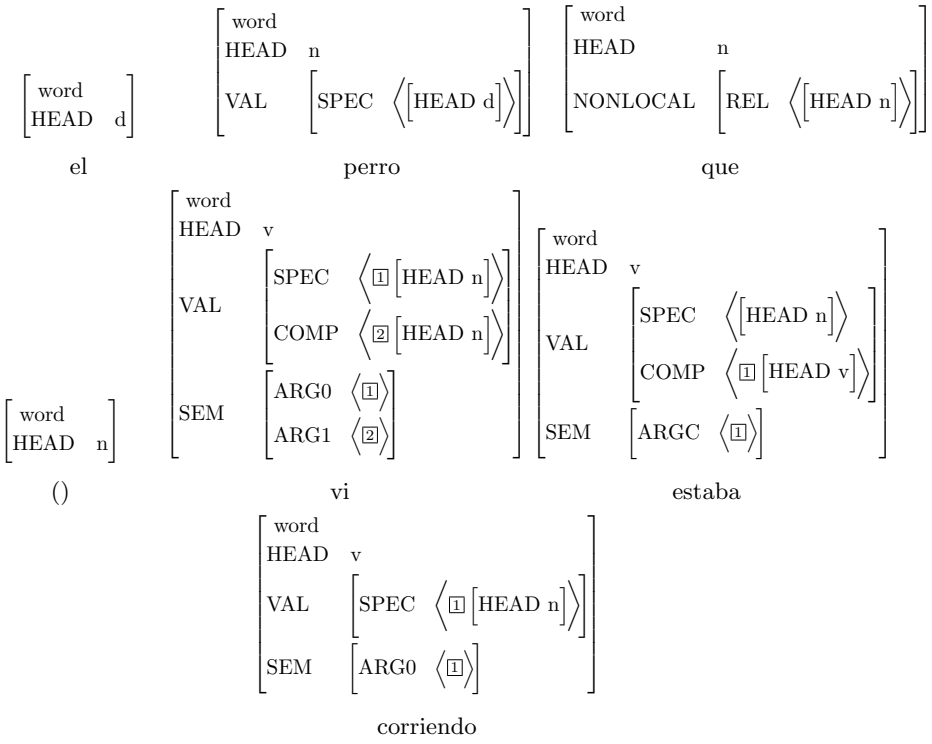


Figure 5.2: HPSG lexical entries for *"El perro que () vi estaba corriendo"*, simplified without morphological features.

of supertags that cannot be combined into a suitable tree, so we would need a backup plan to try to form the best tree we can even if the lexical entries are faulty.

It is also possible that the appropriate lexical entries for the sentence are rare and there are no examples of them in the training corpus. If this is the case, it is hard for a process like a supertagger to come up with new lexical entries it has not seen before.

5.1.2 From trees to lexical entries

On the other hand, we could try an alternative approach that is trying to guess the overall structure of the tree with its rule applications, without knowing the actual entries. After this is ready, we can derive the features of the lexical entries from the rules that are applied in the tree at each step. We know we can do this, because it is the way the corpus is built in the first place (see section 4.6): we labeled each binary tree with the corresponding rule and then inferred the features from that base structure. For example, the process could return a tree like the one in figure 5.3, and we can infer the appropriate features for each

lexical entry from the structure.

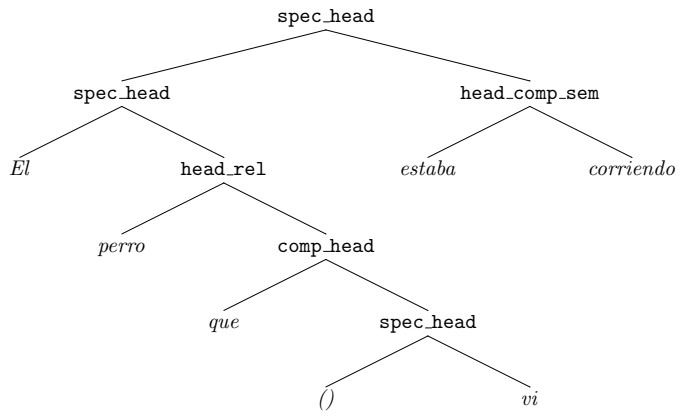


Figure 5.3: Constituency tree for the sentence *"El perro que vi estaba corriendo"* (*"The dog I saw was running"*).

The drawback in this case is that the final trees might end up containing a invalid combination of rules. One example of this is having a tree that applies a `coord_left` rule without its corresponding `coord_right`. It is also possible that we end up with invalid lexical entries. We do not impose many restrictions on the features that could be used for each lexical entry but it could happen, for example, that a `head_rel` rule is applied with a dependent that does not have a relative pronoun. We could either trust that the statistical process learns to infer from the data which situations are invalid and avoid them, or we could use other mechanisms to enforce that the process returns valid trees.

An important advantage of this approach is that it is free to generate any kind of lexical entry it needs to represent the tree, even if it has never seen it in the training set.

5.2 Performance metrics

This section defines the metrics we use to evaluate the performance of the different parsers we built. These are global metrics, which compare the candidate parse tree with the gold standard, the one considered correct from the corpus. Later on we will define other metrics that only focus on some specific aspects.

Consider the following sentence:

(38) *En invierno Juan se irá a Barcelona (In winter Juan will go to Barcelona)*

The parse tree for this sentence in our grammar is shown in figure 5.4. The tree also includes unique identifiers for each word and phrase. The leaves of the tree are the words (w1 through w7) of the sentence and the intermediate nodes are the phrases (p1 through p6).

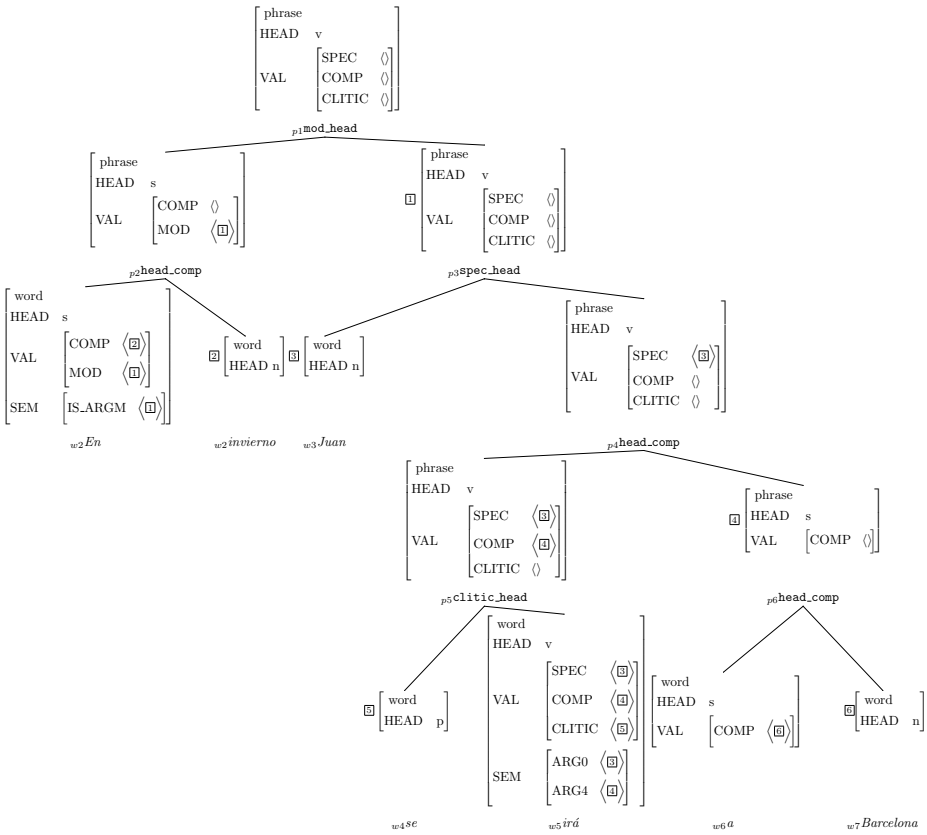


Figure 5.4: Parse tree for the sentence *"En invierno Juan se irá a Barcelona"* (*"In winter Juan will go to Barcelona"*).

5.2.1 Constituency metrics

From the full parse tree shown in figure 5.4 we can extract a constituency tree as the one shown in figure 5.5. The process for generating this tree is straightforward: just replace every leaf in the tree for the corresponding word and every non-leaf for the corresponding rule applied at that node. Notice that not all the information of the original tree can be retrieved from this representation: we could infer all the syntactic features (specifiers, complements, relatives, etc.), but not the corresponding semantic features.

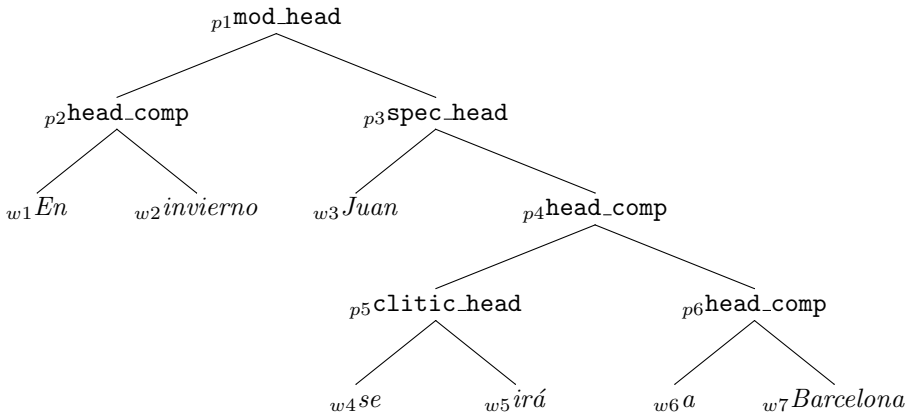


Figure 5.5: Constituency tree for the sentence *"En invierno Juan se irá a Barcelona"* (*"In winter Juan will go to Barcelona"*).

The constituency tree in figure 5.5 also inherits the unique identifiers for each word and phrase of the original tree. Using these identifiers, the tree can also be described as the following list of constituents by taking the sequence of leaves corresponding to each subtree:

- p1: [$_{\texttt{mod\_head}}$ w1 w2 w3 w4 w5 w6 w7]
- p2: [$_{\texttt{head\_comp}}$ w1 w2]
- p3: [$_{\texttt{spec\_head}}$ w3 w4 w5 w6 w7]
- p4: [$_{\texttt{head\_comp}}$ w4 w5 w6 w7]
- p5: [$_{\texttt{clitic\_head}}$ w4 w5]
- p6: [$_{\texttt{head\_comp}}$ w6 w7]

This representation format is useful for defining a metric of comparison between trees. Given two trees in this format, one taken as gold standard and one taken as candidate, we can define the following measures [Jurafsky and Martin, 2014]:

$$\text{Precision: } \frac{|gold\_constituents \cap candidate\_constituents|}{|candidate\_constituents|} \tag{5.1}$$

$$\text{Recall: } \frac{|gold\_constituents \cap candidate\_constituents|}{|gold\_constituents|} \tag{5.2}$$

$$\text{F1 Score: } \frac{2 * precision * recall}{precision + recall} \tag{5.3}$$

Taking in consideration that all the trees used in our process are strictly binary trees, the number of constituents in any tree (gold or candidate) will always be one less than the number of words in the sentence. Because of this, the denominators in equations 5.1 and 5.2 will always have the same value, so both metrics will be the same, and also the F1 score will have the same value as precision and recall. In our results, we will use the constituency F1 score, but it actually represents any of the three values.

The constituency metrics can be defined in two flavors, whether we consider only the structure of the tree or if we also take in consideration the grammar rules applied at each step. If we count the number of constituents considering the rule they apply, it should be considered the *Labeled F1*. Conversely, if we count the constituents without the rule, it should be considered *Unlabeled F1*, which is a more relaxed measure. To distinguish them from other metrics, in this document we will call the labeled constituency F1 the **L-Cons** metric, and its unlabeled version the **U-Cons** metric.

L-Cons

U-Cons

5.2.2 Dependency metrics

Considering that each binary rule defines its children as a head and a dependent, we can transform the tree in figure 5.5 to a format similar to dependency trees by moving the head words up the tree. The process works as follows:

- Select a phrase node that only has words as children, but not other phrases.
- Substitute that phrase node in the tree for its head daughter, which is a word, and merge the children of both nodes (phrase and word).
- Iterate the process until there are no more phrases in the tree.

Figure 5.6 shows this process in action for sample 38. Starting from the constituent tree (5.6a), we first move the head words *"En"*, *"irá"* and *"a"* up one level in the tree (5.6b), then in each iteration only the word *"irá"* will move up the tree until the process converges to the final tree in figure 5.6e.

This tree can be described as a set of three-tuples <dependent word, head word, rule name> in the following way (we include an extra tuple for the root of the sentence, i.e. the word that does not have an outbound dependency):

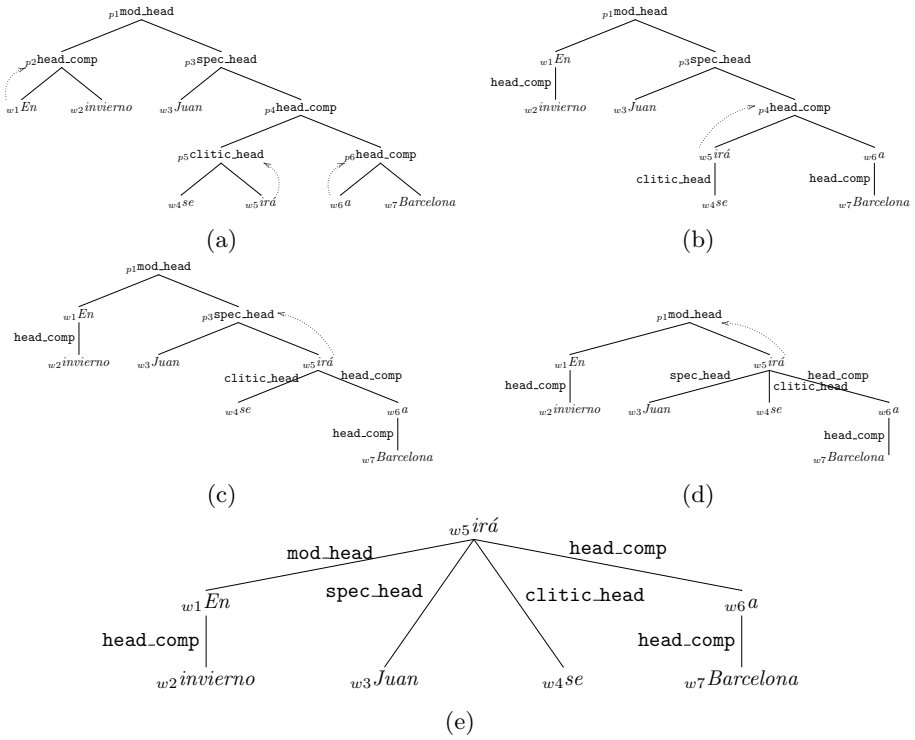


Figure 5.6: Conversion from constituency to dependency tree for the sentence *"En invierno Juan se irá a Barcelona"* (*"In winter Juan will go to Barcelona"*).

- w1, w5, `mod_head`
- w2, w1, `head_comp`
- w3, w5, `spec_head`
- w4, w5, `clitic_head`
- w5, -, `root`
- w6, w5, `head_comp`
- w7, w6, `head_comp`

Notice that this notation is structurally equivalent to a dependency tree notation, but the labels correspond to our grammar rules instead of the usual labels for dependency trees. Every dependency grammar defines its own set of available labels, so this format could be seen as a dependency grammar with a particular set of labels adapted to our needs. Of course it would not be possible to compare the labels with other dependency formats without performing some

kind of transformation, but it is possible to compare different trees annotated in our format.

Given a gold tree and a candidate tree in this format, we can define the metrics corresponding to dependency trees [Buchholz and Marsi, 2006]. The *Labeled Dependency Accuracy* or *Labeled Attachment Score* (LAS) is the proportion of correctly predicted tuples; while the *Unlabeled Dependency Accuracy* or *Unlabeled Attachment Score* (UAS) is the proportion of correctly predicted heads without considering the rule. More formally:

$$\text{LAS: } \frac{|labeled\_tuples\_in\_gold\_tree \cap labeled\_tuples\_in\_candidate\_tree|}{|words|} \tag{5.4}$$

$$\text{UAS: } \frac{|unlabeled\_tuples\_in\_gold\_tree \cap unlabeled\_tuples\_in\_candidate\_tree|}{|words|} \tag{5.5}$$

In order to distinguish them from other metrics, in this document we will call
L-Dep the Labeled Attachment Score the **L-Dep** metric and the Unlabeled Attachment
U-Dep Score the **U-Dep** metric.

The constituency metrics L-Cons and U-Cons are more strict than the dependency metrics L-Dep and U-Dep defined in this section. This can be seen in the following example: Consider the candidate parse tree in figure 5.7. This tree is different from the one in figure 5.5 in that in this case the subject *"Juan"* is applied to the verb before applying the complement *"a Barcelona"*.

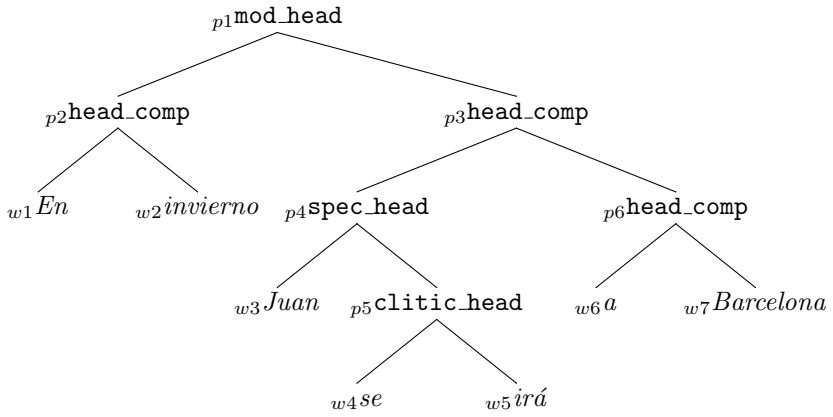


Figure 5.7: Constituency tree of a parser candidate for the sentence *"En invierno Juan se irá a Barcelona"* (*"In winter Juan will go to Barcelona"*).

For this candidate tree, the list of constituents is the following:

- p1: [$_{\texttt{mod\_head}}$ w1 w2 w3 w4 w5 w6 w7]
- p2: [$_{\texttt{head\_comp}}$ w1 w2]
- p3: [$_{\texttt{head\_comp}}$ w3 w4 w5 w6 w7]
- p4: [$_{\texttt{spec\_head}}$ w3 w4 w5]
- p5: [$_{\texttt{clitic\_head}}$ w4 w5]
- p6: [$_{\texttt{head\_comp}}$ w6 w7]

This constituent tree presents a violation with respect to the standard behavior of HPSG: the complements should be attached to the head before the specifier. However, it is a typical error that a parser might make, and from the point of view of the attachments of arguments to the respective heads it behaves almost as well as the correct tree.

Notice that in this case, only five out of the six expected constituents are present in the candidate tree, and only four of them have the appropriate rule. This means that U-Cons is 0.83 and L-Cons is 0.67 for this example. However, if we transform the constituency tree to dependencies using the process defined above, we will arrive to the exact same tree as 5.6e, which means the U-Dep and L-Dep in this case are both 1.0. This gap between the two metrics could become much higher for longer sentences, because increasing the number of constituents will result in exponentially more possibilities in the order of application of the rules that represent the same dependency tree.

5.2.3 SRL metrics

So far we have only dealt with the evaluation of the structural syntactic information contained in the parse trees, whether constituents or dependencies, but we left out their semantic information. Given that our semantic representation is SRL, we could use standard SRL metrics for this. In this work we base our metrics on the ones used in CoNLL 2009 shared task [Hajič et al., 2009]. They propose building a semantic dependency graph that contains an arc when there is a semantic dependency between an argument and a predicate, labeled with the argument type, and another arc between the predicate and the root, labeled with the predicate type. Both the argument and the predicate are represented by their heads in the dependency tree. Then they compare the arcs in a gold and candidate tree using precision, recall and F1 score.

In our work we are not trying to predict the predicate types, so we discard the arcs corresponding to these types and only keep the arcs between arguments and predicates. The dependency tree with the semantic annotations corresponding to the tree defined in 5.4 is shown in figure 5.8. Notice that the semantic arguments that are introduced by endocentric semantic features (like `IS_ARGM`)

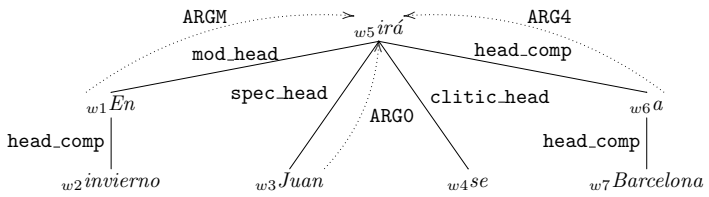


Figure 5.8: Dependency tree with semantic annotations for the sentence *"En invierno Juan se irá a Barcelona"* (*"In winter Juan will go to Barcelona"*).

are represented as their exocentric versions in this representation, so the relation between *"irá"* and *"En invierno"* becomes `ARGM`.

From this tree we can extract the following set of semantic dependencies:

- w1, w5, `ARGM`
- w3, w5, `ARG0`
- w6, w5, `ARG4`

Using the semantic dependencies, the SRL metrics between a gold and a candidate tree are defined in the following way:

$$\text{Precision: } \frac{|gold\_sem\_dependencies \cap candidate\_sem\_dependencies|}{|candidate\_sem\_dependencies|} \tag{5.6}$$

$$\text{Recall: } \frac{|gold\_sem\_dependencies \cap candidate\_sem\_dependencies|}{|gold\_sem\_dependencies|} \tag{5.7}$$

$$\text{F1 Score: } \frac{2 * precision * recall}{precision + recall} \tag{5.8}$$

In this work we will mainly refer to the F1 score as the main metric for SRL. We will define two versions of this metric, one of them considering a match if the semantic dependency is predicted correctly together with its argument type (the Labeled F1 Score, which we call **L-SRL**), and an unlabeled version of this metric that considers a match for the pair dependency-predicate regardless of the argument type (the Unlabeled F1 Score, which we call **U-SRL**)[1].

L-SRL

U-SRL

[1]SRL systems are usually evaluated using the labeled metric, but for completeness we include the unlabeled metric in our results as well.

5.3 Bottom-up Baseline

To begin with our parsing experiments, we first designed a very simple parsing strategy that we use as a baseline in order to compare to more complex approaches. This simplest strategy uses a greedy approach to build the parse tree in a bottom-up way. Consider the following example:

(39) *El niño come una manzana roja (The kid eats a red apple)*

The expected constituency tree for sentence 39 in our grammar is shown in figure 5.9.

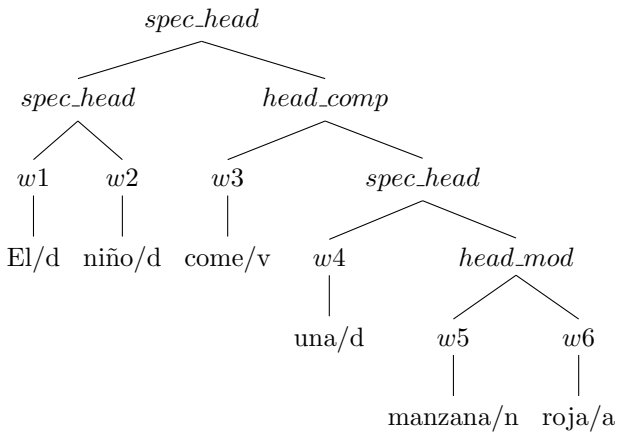


Figure 5.9: Simplified constituents tree for *"El niño come una manzana roja"* (*"The kid eats a red apple""*) without feature structures and semantic role label information.

Given the way the grammar works, we can be sure that in the expected tree there are at least two consecutive words of the original sentence that should be combined together to form a phrase. The aim of the bottom-up baseline parser is to find a suitable pair of consecutive words to combine, select the appropriate rule, and substitute the words for a phrase. Then it will proceed iteratively until all words are structured into phrases. Let us see how it works in this example.

First of all, there are two possible phrases that can be extracted in this case:

[El niño] come una [manzana roja]

If the process extracts any of those two phrases, the final tree will be the same. The process might decide to extract, for example, the phrase *[manzana roja]* and decide that the rule to apply is `head_mod`. Then it will substitute the pair of words for the syntactic head of the phrase, leaving the following:

El niño come una manzana

Now there are two more options to reduce: *[El niño]* and *[una manzana]*. The process might decide to extract *[El niño]* and label it with the `spec_head` rule. After reducing the phrase, the rest of the sentence looks like the following:

niño come una manzana

From now on, there is only one way of extracting phrases and reducing the tree in order to get the gold parse tree. The process must first extract *[una manzana]* as `spec_head` and substitute it with *manzana*, then it must extract *[come manzana]* as `head_comp` (leaving *come*) and finally extract *[niño come]* as `spec_head`. With this sequence of reductions, the process yields the parse tree shown in figure 5.9. We will call this sequence of steps the **parsing history** for the sentence.

Parsing history

What the process needs to learn is how to decide which pair of words is the most suitable to extract on the next step, and which rule it should apply. We created different statistical models for this, which will be explained in increasing order of complexity in the following sections.

5.3.1 Bilexical comparison

The easiest way to make the comparison is creating a scoring function that assigns a higher score to pairs of words that have been seen together in the training corpus, and less score, to the pairs that have not been seen together.

Consider the example 39 above and the new example 40

(40) *La manzana verde es rica (The green apple is tasty)*

Let us walk through the score definition using these examples. First of all, write down the parsing history for both sentences 39 and 40, as shown in table 5.1.

Example	Step	Text
39	1	*[el niño]$_{spec\_head}$ come una [manzana roja]$_{head\_mod}$*
	2	*niño come [una manzana]$_{spec\_head}$*
	3	*niño [come manzana]$_{head\_comp}$*
	4	*[niño come]$_{spec\_head}$*
40	1	*la [manzana verde]$_{spec\_head}$ [es rica]$_{head\_comp}$*
	2	*[la manzana]$_{spec\_head}$ es*
	3	*[manzana es]$_{spec\_head}$*

Table 5.1: Parsing history for sentences 39 and 40.

The objective is to build a table that indicates, for every pair of consecutive words, if they should be reduced and which rule should be applied. In this first approach we just count all pairs of consecutive words and the rule applied for each one, considering that if two consecutive words are not reduced in a step, the corresponding rule is a fictitious label `none`. Let us assume for a moment

that the only possible rules are `spec_head`, `head_comp`, `head_mod`, and `none`, then table 5.2 shows the normalized counts for the word pairs in our examples.

Left	Right	`none`	`spec_head`	`head_comp`	`head_mod`
come	manzana	0	0	1	0
come	una	1	0	0	0
el	niño	0	1	0	0
es	rica	0	0	1	0
la	manzana	0.5	0.5	0	0
manzana	es	0.5	0.5	0	0
manzana	roja	0	0	0	1
manzana	verde	0	0	0	1
niño	come	0.75	0.25	0	0
una	manzana	0.5	0.5	0	0
verde	es	1	0	0	0

Table 5.2: Normalized pair counts.

Now let us try to parse a new sentence using the scores defined by this table. Consider example sentence 41, which is a combination of the previous sentences.

(41) *El niño come una manzana verde (The kid eats a green apple)*

First of all, we want to know which pair of consecutive words should be reduced. In order to do that, we try to find the score associated to `none` for each pair of words:

Step 1										
Words:	el		niño		come		una		manzana	verde
Score:		0		0.75		1		0.5		0

The pairs of words that are least likely to be labeled as `none` (the smallest score) are *"el niño"* and *"manzana verde"*. We can take the first pair (*"el niño"*) to reduce, and look up in the table what is the most likely rule to apply, in this case `spec_head`. We reduce this pair and substitute the word for the head of the phrase (*"niño"*), and we iterate the process:

Step 2									
Words:	niño		come		una		manzana		verde
Score:		0.75		1		0.5		0	

In this second step we reduce *"manzana verde"* using the rule `head_mod`, and leave *"manzana"* for iterating the process:

Step 3								
Words:	niño		come		una		manzana	
Score:		0.75		1		0.5		

In the third step we reduce *"una manzana"* using the rule `spec_head`.

Step 4			
Words:	niño	come	manzana
Score:	0.75	1	

Thus, in the fourth step we reduce *"come manzana"* with rule `head_comp` and substitute it for *"come"*.

Step 5		
Words:	niño	come
Score:	0.75	

In the fifth and final step it is not even necessary to calculate the score of `none`, as there will only be two words, but we must look up in the table to see what is the most likely rule to apply. In this case it is `spec_head`. The resulting tree is shown in figure 5.10.

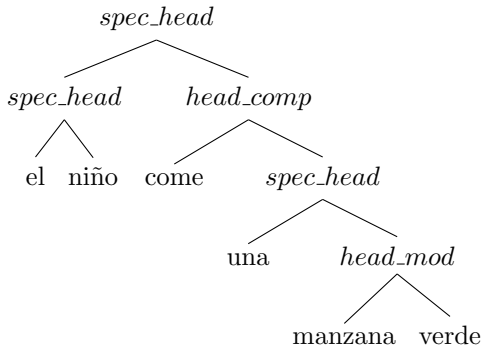


Figure 5.10: Constituents tree for *"El niño come una manzana verde"* (*"The kid eats a green apple"*) after the parsing process.

This simple process can be extended to the whole training corpus and considering all possible rules, but it is of course far from perfect. The first problem we run into is what to do if a word or a pair of consecutive words are not found in the training corpus. We designed a simple back-off mechanism for handling this situation that uses the parts of speech of the words, which must be indicated in the sentence at parse time.

The back-off mechanism implies adding new fictitious words to the corpus that represent an unknown word for each part of speech. New entries are added to the table that represent the behavior of this unknown words with respect to other words. Each pair of words <word1, word2> with parts of speech <pos1, pos2> in the corpus counts as if it was these four entries:

- <word1, word2>→ `rule`
- <word1, pos2>→ `rule`

- <pos1, word2>→ `rule`
- <pos1, pos2>→ `rule`

For example, the words *"el niño"* from step 1 of the parsing history of sentence 39 would provide the following counts:

- <el, niño>→ `spec_head`
- <el, unk_N>→ `spec_head`
- <unk_D, niño>→ `spec_head`
- <unk_D, unk_N>→ `spec_head`

Left	Right	none	spec_head	head_comp	head_mod
come	manzana	0	0	1	0
come	una	1	0	0	0
come	unk_D	1	0	0	0
come	unk_N	0	0	1	0
el	niño	0	1	0	0
el	unk_N	0	1	0	0
es	rica	0	0	1	0
es	unk_A	0	0	1	0
la	manzana	0.5	0.5	0	0
la	unk_N	0.5	0.5	0	0
manzana	es	0.5	0.5	0	0
manzana	roja	0	0	0	1
manzana	unk_A	0	0	0	1
manzana	unk_V	1	0	0	0
manzana	verde	0	0	0	1
niño	come	0.75	0.25	0	0
niño	unk_V	0.75	0.25	0	0
una	manzana	0.5	0.5	0	0
una	unk_N	0.5	0.5	0	0
unk_A	es	1	0	0	0
unk_A	unk_V	1	0	0	0
unk_D	manzana	0.5	0.5	0	0
unk_D	niño	0	1	0	0
unk_D	unk_N	0.4	0.6	0	0
unk_N	come	0.75	0.25	0	0
unk_N	es	0.5	0.5	0	0
unk_N	roja	0	0	0	1
unk_N	unk_A	0	0	0	1
unk_N	unk_V	0.667	0.333	0	0
unk_N	verde	0	0	0	1
unk_V	manzana	0	0	1	0
unk_V	rica	0	0	1	0
unk_V	una	1	0	0	0
unk_V	unk_A	0	0	1	0
unk_V	unk_D	1	0	0	0
unk_V	unk_N	0	0	1	0
verde	es	1	0	0	0
verde	unk_V	1	0	0	0

Table 5.3: Normalized pair counts with backoff.

After counting all entries in this fashion, the normalized counts for all pairs of words (and unknown words) looks like table 5.3. Now the score for applying a rule to a pair of words can be defined as a linear combination of the scores applied to the words plus the different combinations of words and parts of speech:

$$s_{rule} = \alpha_1 s(w1, w2) + \alpha_2 s(w1, p2) + \alpha_3 s(p1, w2) + \alpha_4 s(p1, p2) \qquad (5.9)$$

We used the development corpus to empirically determine the most suitable combinations of weights, and we found the best score in this case is: $\alpha_1 = 0.7, \alpha_2 = \alpha_3 = \alpha_4 = 0.1$. By default, if a combination of words is not in the table, the standard score for the pair <word1, word2> is 0 for all rules except for `none`, whose value is 1.

Let us try to parse a sentence with unknown words using this process.

(42) *El niño ve la manzana (The kid sees the apple)*

The parsing process for sentence 42 would be like the following:

Step 1					
Words:	el	niño	ve	la	manzana
POS:	D	N	V	D	N
Score:	0.04	0.94	1	0.49	
Action:	Reduce *"el niño"* with rule `spec_head`				
Step 2					
Words:	niño	ve	la	manzana	
POS:	N	V	D	N	
Score:	0.94	1	0.49		
Action:	Reduce *"la manzana"* with rule `spec_head`				
Step 3					
Words:	niño	ve	manzana		
POS:	N	V	N		
Score:	0.94	0.8			
Action:	Reduce *"ve manzana"* with rule `head_comp`				
Step 4					
Words:	niño	ve			
POS:	N	V			
Score:	0.94				
Action:	Reduce *"niño ve"* with rule `spec_head`				

Table 5.4 shows the performance of this approach over the validation corpus for the global performance metrics defined in section 5.2. From now on, we will refer to this approach as **Bottom-up Bilex Context 0**.

	U-Cons	L-Cons	U-Dep	L-Dep
Bottom-up Bilex Context 0	49.19	38.40	63.37	59.92

Table 5.4: Performance for the bilexical comparison baseline over the development corpus.

5.3.2 Bilexical comparison with context

One problem with this approach is that it is too local: the score for a pair of words only depends on the words but not on the context. This might lead the

parser to make greedy decisions early on that hinder the parsing performance. For example for a phrase like *"la casa de madera"* (*"the wooden house"*) the parser might choose to reduce the pair *"casa de"* before it reduced the pair *"de madera"*, just because one of them might be more frequent in the corpus.

In order to mitigate this problem, we extend the first approach by adding one word of context to the left and one to the right of the pair of words (thus we will call this approach **Bottom-up Bilex Context 1**). So the new scoring function will try to obtain the score and the rule to apply to the pair *<word1, word2>* given that the word *context1* is on the left and the word *context2* is on the right.

For this to work properly, we add two extra tokens (`<none>`) to each sentence marking the beginning and the end of the sentence, so that the first and last words of the sentence have appropriate context tokens.

Tabe 5.5 illustrates what happens to *"la casa de madera"* in this approach compared to the previous one. The scores in the table are the real scores calculated from the training corpus. The heuristic without context has an error on the first step that carries on the rest of the derivation, while in this case the heuristic with context handles the situation correctly. Notice that on the left side the score assigned to the bigram *"la casa"* is constant through the steps, but on the right side it changes when the trailing context changes.

	Bilexical wihtout context				Bilexical with context			
Step 1								
Words:	la	casa	de	madera	la	casa	de	madera
POS:	D	N	S	N	D	N	S	N
Score:	0.723	0.693	0.712		0.839	0.815	0.765	
Action:	Reduce *"casa de"* with rule `head_mod`				Reduce *"de madera"* with rule `head_comp`			
Step 2								
Words:	la	casa	madera		la	casa	de	
POS:	D	N	N		D	N	S	
Score:	0.723	0.082			0.839	0.775		
Action:	Reduce *"casa madera"* with rule `head_mod`				Reduce *"casa de"* with rule `head_mod`			
Step 3								
Words:	la	casa			la	casa		
POS:	D	N			D	N		
Score:	0.723				0.702			
Action:	Reduce *"la casa"* with rule `spec_head`				Reduce *"la casa"* with rule `spec_head`			

Table 5.5: Comparison of the bottom-up parsing strategies for *"la casa de madera"* (*"the wooden house"*) with and without using context.

In this case, handling out of vocabulary words becomes more complicated, as we now have several combinations of known and unknown words that should be taken in consideration. We performed a series of experiments tuning the weights of the different combination of words against the development corpus and found out that the best configuration was using a weight of 2/5 for the perfect match of words, and 1/25 to every other possible match (compare to the weights found for equation 5.9).

Table 5.6 shows the global metrics over the development corpus for this heuristic. Interestingly, the approach significantly improves the constituents metrics (both labeled and unlabeled). However, the dependency metrics vary much less: the unlabeled dependency metric improves a little, and the labeled dependency metric remains almost the same, even falling a little. This might indicate that the heuristic with context helps the system learn the order of application of the rules better, but it is not that significant for finding the heads of the words.

	U-Cons	L-Cons	U-Dep	L-Dep
Bottom-up Bilex Context 1	61.13	47.17	65.36	59.81

Table 5.6: Performance for the bilexical comparison with context baseline over the development corpus.

We could expand this approach to have even more words of context (e.g. two to the left and two to the right), but in this case the counts in the corpus start to be very sparse, so we cannot calculate the probabilities accurately. There is, however, another way of expanding the context that we will see in the following section.

5.3.3 Multilayer Perceptron

In order to expand the number of context tokens without losing generality due to sparsity, we can use word embeddings. Word embeddings have the property that semantically related words are nearer in the embeddings space than unrelated words. So if we train the model with sentence 39 *"el niño come una manzana roja"*, it would tend to generalize to some other concepts that could be close in the embeddings space such as other fruits, colors, or verbs related to eating. Ideally, this would imply that even with few examples of each concept the model could generalize better.

The main idea for this heuristic is analogous to the bottom-up approach we have described in this section, but in this case instead of a table holding the scores for the different combinations of words, we will train a Multilayer Perceptron (see section 2.6.2) neural network that will model the scoring function we want.

The architecture of the network, as shown in figure 5.11, is the following:

- Input layer: One input for each word considered. We will call this size N.
- Embeddings layer: This layer outputs 300 dimensions for each word. The size of the output is $N \cdot 300$.
- Dense layers: Two dense layers with `relu` activation that take the $N \cdot 300$ parameters as input.
- Output layer: A dense layer with `softmax` activation that has 14 output units (one for each rule and one for `none`).

This strategy is inspired in the work of [Socher et al., 2013], although our approach is simpler. Instead of using the network as a recursive unit that outputs an embedding representing the combination of two elements, our network will only be used to decide which rule to apply, and the embedding of the combination will be the embedding of the word that corresponds to the head word.

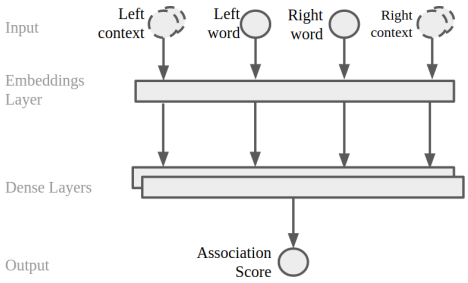


Figure 5.11: Architecture of the MLP neural network. The dashed lines represent the variable number of words in context.

The training data for the network corresponds to the same training points described in section 5.3.1 and shown in table 5.1. There is one training point for each consecutive pair of words in the training history. However, using the embeddings allows us to extend the context we present to the network, so we tried two versions of the architecture:

- Context 1: using the two words plus one word to the left and one to the right for context (this implies $N = 4$ in the network architecture). After tuning with the development corpus, the best model for Context 1 has 400 units in the first layer, 100 units in the second layer, and dropout 0.2.
- Context 2: using the two words plus two words to the left and two to the right for context (this implies $N = 6$ in the network architecture). After tuning with the development corpus, the best model for Context 2 has 450 units in the first layer, 150 units in the second layer, and dropout 0.1.

The words are represented by word embeddings trained from a six billion words Spanish corpus [Azzinnari and Martínez, 2016] using the `word2vec` implementation of the `gensim` library [Řehůřek and Sojka, 2010]. The word embeddings set has 1,146,242 vectors of dimension 300. But even with this large collection of embeddings, there could be words in the corpus (and later on in a real parsing scenario) that are not in the embeddings table. In order to handle these out of vocabulary words, we created a set of different tokens that represent unknown words, one for each POS. We listed all the combinations of parts of speech with their corresponding morphological attributes in the training corpus and calculated the most frequent word for each combination. Then we found the embeddding corresponding to that word in our collection, and created an embedding for representing an unknown word for each POS using that an unknown token for each POS tag (considering morphological information) and use

the vector corresponding to the most frequent word for that POS. For example, the most frequent word for the POS `NP0000L` (proper noun of type location) is *"España"* (*"Spain"*), so the embeddding for *"España"* would be used in lieu of the embeddings of other locations if they are not found in the vocabulary. This is of course far from perfect, as all the unknown locations will be treated as *"España"*, but is nonetheless better than using null or random embeddings for those cases, as the embedding for *"España"* is more likely to share some characteristics with the other countries or locations embeddings.

Table 5.7 shows the performance of these approaches (we will call them **Bottom-up MLP Context 1** and **Bottom-up MLP Context 2**) over the development corpus. Notice that even the context 1 approach improves over the bilexical comparison with context approach, which could be explained by the powerful generalization capabilities of word embeddings and MLP.

	U-Cons	L-Cons	U-Dep	L-Dep
Bottom-up MLP Context 1	70.53	61.46	79.96	75.17
Bottom-up MLP Context 2	74.18	65.17	81.57	76.70

Table 5.7: Performance for the MLP baselines over the development corpus.

5.3.4 SRL baseline

So far these baselines have only focused in the syntactic aspect of the parsing process, but our model also considers semantic information in the form of SRL annotations for predicates. We implemented a simple rule based heuristic for attempting SRL on top of the result of the purely syntactic parsers.

The heuristic first calculates the valence features for each node in the tree based on the rules selected during the parsing process. For sample sentence 39 the resulting tree is the one shown in figure 5.12. This process is straightforward (see section 4.6): we start with the root and traverse each daughter collecting syntactic features based on the rule applied at each step (e.g. `head_spec` and `spec_head` create a `SPEC` feature, while `head_mod` and `mod_head` create a `MOD` feature). The process traverses the tree recursively and stops at each leaf.

Next we traverse the tree again and use the following rules for calculating the semantic features of each leaf:

- If the word is a verb and has a `SPEC` feature, coindex it with its `ARG0` feature.
- If the word is a verb, noun or adjective and has a `COMP` feature, coindex all its complements with its `ARG1` feature.

This is a really naïve heuristic for calculating SRL, but it helps us complete the baseline parsing process and sets the ground for comparing these heuristics to more advances models. The performance of this SRL heuristic over the development corpus for the different baselines is shown in table 5.8.

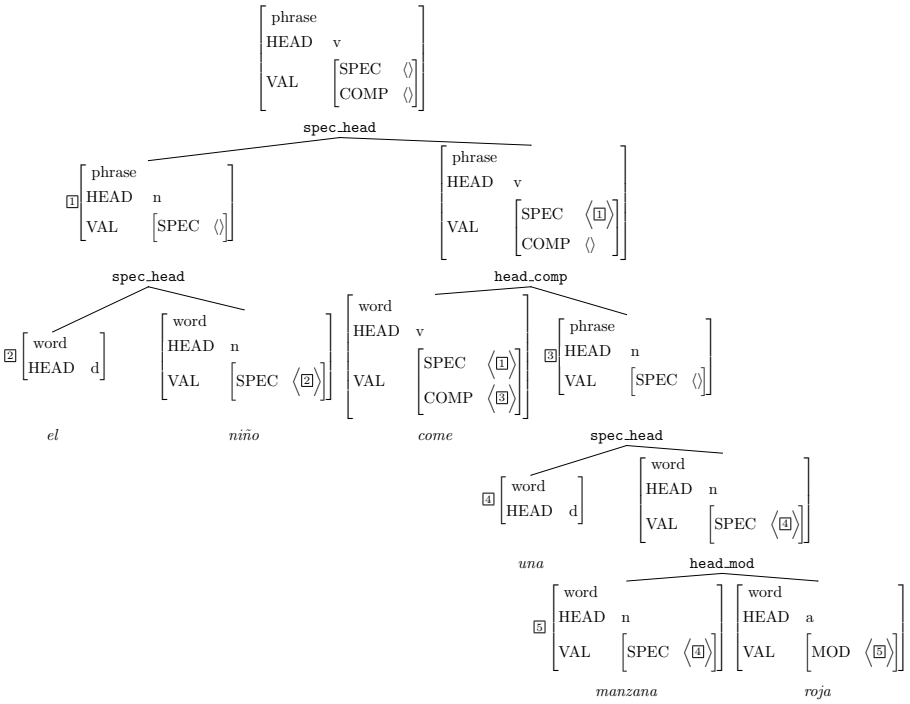


Figure 5.12: Tree with syntactic features for the sentence *"el niño come una manzana roja"* (*"the kid eats a red apple"*).

	U-SRL	L-SRL
Bottom-up Bilex Context 0	59.14	45.72
Bottom-up Bilex Context 1	58.93	44.58
Bottom-up MLP Context 1	65.56	49.21
Bottom-up MLP Context 2	67.35	50.41

Table 5.8: SRL F1 score performance for the different baselines over the development corpus.

5.4 CKY with Supertags

The baseline processes described so far have some very important drawbacks. First of all, once the next pair of words to reduce has been decided, there is no turning back to analyze other options. That is because the process is greedy. This has the advantage of being fast, as each option is only considered once, but also the disadvantage that errors committed early in the process condition the rest of the parsing process and could easily cascade into other errors.

Another problem with the approaches above is that they only consider the information conveyed by words, but not other information that would be available in a HPSG approach. Furthermore, once we decided which pair of words to reduce, we simply substitute the pair for the head word according to the selected rule. This implies forgetting the derivation history so far, information that could be exploited in order to improve the rest of the process.

In order to deal with these problems, we implemented a more standard strategy that has been used for parsing HPSG [Matsuzaki et al., 2007] and other grammar formalisms. Given our grammar is binarized, it is possible to use the well known CKY algorithm for parsing (see section 2.2.1). Instead of using a greedy process, CKY is a dynamic programming algorithm that keeps a chart of partially parsed trees and analyzes all possible combinations bottom-up in polynomial time. It is possible that many valid trees are generated in the process, so we also need a probabilistic model for determining which of the trees is the most likely one.

The simplest way of implementing this CKY process would be directly using the unification process: at each step we try to apply every rule (unifying the rule with the left and right subtrees), and we add the tree to the chart if the unification is sound. This has an important shortcoming: the unification process is so slow that parsing with this strategy becomes impossible. So in order to make this work in a reasonable time, we implemented a speed up strategy based on the use of supertags: We designed a set of supertags based on the grammar categories that contain the necessary information to infer the possible rules to apply given a pair of supertags.

The supertags are used in two ways in our process: on the one hand we implemented a standard approach to supertagging meant to disambiguate the possible categories to apply to a word, and on the other hand we will use the same tags to speed up the calculation of the possible rules to apply.

5.4.1 Supertags

Our definition of supertags tries to represent the way a word is being used in the context of a sentence. It describes the feature slots the word has to fill and in which positions with respect to the current word. These supertags are designed to be very expressive in terms of the description of the word, so the parsing process becomes as unambiguous as possible.

Consider again the sample sentence 39:

El niño come una manzana roja.
(The kid eats a red apple.)

The main verb of the sentence (*come*) has a noun phrase specifier to its left (*El niño*) and a noun phrase complement to its right (*una manzana roja*). This will be the information conveyed by the supertag for *come*, as shown below:

come/`v-sna0-x-cna1`

The leftmost character is the part of speech of the word. The rest of the supertag is a description of the use of the word in that context, where `x` represents the position of the word. In this case, `sna0-x-cna1` encodes the following information: the word will have a noun phrase acting as specifier on its left (`sn`) coindexed with `ARG0`, plus a noun phrase acting as a complement on its right (`cn`) coindexed with `ARG1`.

The corresponding supertags for the whole sentence are the following:

El/`d-x` *niño*/`n-sd-x` *come*/`v-sna0-x-cna1` *una*/`d-x` *manzana*/`n-sd-x`
roja/`a-mn-x` *./*`f-x`

Notice that when the word has no syntactic valence on its own (such as *una*/`d-x`), its supertag is simply *POS*`-x`. The supertags have the following syntax:

supertag = *POS*`-`*features_list*`-x-`*features_list*
features_list = *feature_def* (`-` *feature_def*)`*`
feature_def = *feature POS* `[` *arg_def* `]`
feature = `s` | `c` | `t` | `m` | `r` | `e`
POS = *any valid part of speech*
arg_def = `(` `a` | `i` `)` `(` `0` | `1` | `2` | `3` | `4` | `m` | `l` `)`

The supertags contain the same information that the feature structure for the word contains. For example, the supertag for the adjective *roja* indicates it is expected to modify a noun phrase on its left: `a-mn-x`. In this case, the head of the resulting structure will not be the adjective, but the noun phrase (because `MOD` is an exocentric feature). On the other hand, the word *manzana* has the supertag `n-sd-x`, which means it expects a determiner to act as specifier, and the resulting phrase will have the noun as its head (because `SPEC` is an endocentric feature). The supertag markers corresponding to the different features can be classified in the following way:

- Endocentric feature markers:
  - `s` - `SPEC` - Specifier
  - `c` - `COMP` - Complement
  - `t` - `CLITIC` - Clitic pronoun

- Exocentric feature markers:
  - `m` - `MOD` - Modifier
  - `r` - `REL` - Relative marker
  - `e` - `COORD_LEFT` or `COORD_RIGHT` - Coordinated element

At the same time, we can consider the argument markers `a` as endocentric and `i` as exocentric, like the corresponding semantic argument features.

5.4.2 Expressing the grammar rules using supertags

The possible rules to apply to a pair of words will depend on the supertags associated to those words. Once the words are combined following certain rule, the resulting phrase will have a new supertag that could be used to continue the parsing process. The supertag structure is simple enough so that all the rules defined in our grammar can be expressed in terms of regular expressions (with capture groups) over pairs of supertags, which is much faster than the unification method, but at the same time expressive enough so that the invalid combinations are filtered out. Consider the following example:

$$\texttt{P1-L1x-cP2R1 + P2-x => P1-L1xR1 , head\_comp}$$
$$\texttt{P1,P2} \in \textit{POS}$$
$$\texttt{L1,L2} \in \texttt{features\_list}$$

The expression on the left has the form `P1-L1x-cP2R1`, which means:

- `P1` is its part of speech.
- It has some pending features (`L1`) on the left.
- It expects a complement of type `P2` on the right.
- It has some more pending features (`R1`) on the right.

The expression on the right has the form `P2-x`, which means its part of speech is compatible with the expression on the left, and it has no pending features. In that case, the rule `head_comp` can be applied and the resulting supertag will be the same as the left side supertag, removing the complement feature (`P1-L1xR1`).

Using these regular expressions, it is possible to write all the grammar rules. The number of regular expressions needed to cover each grammar rule is shown in table 5.9. The most complex rule to cover is `head_comp_sem`, the rule used to build verb phrases that percolate the arguments of a complement to the verb head. Five expressions are needed to cover the different scenarios in which this could happen, considering the presence or absence of subject and the different left and right scenarios.

Rule	Number of r.e.
head_punct	1
punct_head	1
mod_head	2
spec_head	2
comp_head	2
head_spec	1
head_comp	2
head_comp_sem	5
head_mod	1
clitic_head	1
head_rel	1
coord_left	1
coord_right	1

Table 5.9: Number of regular expressions used for each grammar rule.

5.4.3 Probabilistic Model

We created a probabilistic model for this grammar in a way similar to the model of a probabilistic context-free grammar. In this type of grammars (see section 2.1.2), the probabilities of all the rules with the same left hand side (the same non terminal) must add to 1. The probability of a derivation tree is the product of the probabilities of applying each one of the rules.

One way of generating a probabilistic model for this grammar would be considering each possible supertag as a potential non-terminal, and all the ways of combining it to other supertags to form the different rules for that non-terminal. The problem with this approach is that the number of supertags is very large. There are potentially infinite possible supertags if we consider that the number of complements is unbounded, though in real examples one should not expect more than four or five complements, even so the number of combinations is huge. In our training corpus there are 4,146 different supertags. As the number of possible supertags is so big, the number of times the combination of a particular pair of supertags appears in the corpus is consequently very low (sparsity problem). Because of this, we need a way of reducing the number of non terminals.

In this work, we propose two ways of creating simpler models for abstracting the non terminals:

Abstract tags model 1: Reduce a supertag to only the POS of the word followed by a set of flags that indicate if a feature is expected or not (regardless of how many copies of the feature the supertag has, in which positions they are, what POS they have and their SRL information).

For example, according to model 1, the word *daba*/`v-sna0-x-cna1-csa2` would have an abstract tag `vcs`, which means it is a verb that expects comple-

ments and specifier, but not the finer grained information. A verb that expects the specifier and only one complement would have exactly the same abstract tag.

Abstract tags model 2: Reduce a supertag to only the POS of the word followed by a set of flags that indicate the expected features and corresponding parts of speech (regardless of their position with respect to the word and SRL information).

For example, according to model 2, the word *daba*/`v-sna0-x-cna1-csa2` would have an abstract tag `vcncssn`, which means it is a verb that expects a nominal complement, a prepositional complement and a nominal specifier.

The probabilistic model for our CKY parsing process uses an average of the probabilities calculated over these two abstract tag models, and assigns a very small probability to unseen tag combinations in order to avoid giving rare examples zero probability.

Notice that the abstract tags have no effect in the process of determining the possible rules to apply (the full supertag is used for that), but only for estimating the probability of application of a rule.

5.4.4 Supertagger

The CKY algorithm relies on knowing the exact categories of the words to find the valid rules to apply, but when parsing a sentence from scratch that information would not be available. As the number of possible categories per word is large, we trained a supertagger for calculating the most suitable supertags given a sentence. The supertagger uses as information the words and POS-tags for the sentence and returns a sequence of supertags.

This supertagger is built using stacked LSTM neural networks. The structure of the network (after tuning against the development corpus) is the following (see figure 5.13):

- Input: Words and corresponding POS-tags
- Embeddings layer: word embeddings of size 300, POS embeddings of size 5.
- LSTM layers: Three layers of stacked bi-directional LSTMs with size 450 in each direction and activation `tanh`.
- Dense layer: A fully connected layer of size 300 and activation `tanh`.
- Output: Layer that selects one out of 4,146 possible supertags with activation `softmax`.

Using this network, we get a 89.1% accuracy over the development corpus for the top selected tag, 94.3% when choosing the top two tags, and 96.0% when choosing the top three tags.

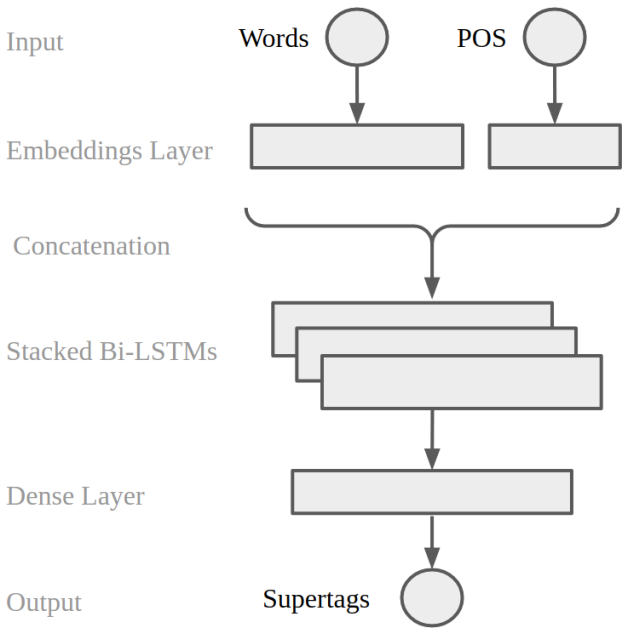


Figure 5.13: Architecture of the neural network for supertagging.

5.4.5 CKY Parser

We implemented the CKY algorithm adapted to our grammar rules and our probabilistic model. The number of possible rule combinations grows very steeply with the size of input, and this makes the parsing process very time consuming for long sentences. Because of this, we limit the number of derivations to explore in each cell of the CKY matrix to a maximum of n, taking only the partial derivations that yield the highest probability for the same combination of words. Initial experiments over the development corpus indicated that the best combination of speed and performance could be achieved using $n = 5$, so we used that number in the rest of our experiments.

However, as the performance of the supertagger is not perfect, the sequence of supertags selected might be invalid for forming a tree according to the grammar rules. To mitigate this problem, if a tree is not found we enable two fallback rules with very low probability (`head_none` and `none_head`) that combine two arbitrary nodes and take either the left one or the right one as heads. These rules guarantee that a tree will be found, but they make the process much slower as many more subtrees are tried during parsing. When the fallback rules are enabled, we use $n = 2$ instead of $n = 5$ so the process becomes faster.

Table 5.10 shows the results of using the CKY process over the development corpus, both for the unrealistic ideal scenario of knowing the correct supertags for the sentence (**CKY Gold tags**), and the more realistic scenario of using the supertagger prior to executing the CKY algorithm (**CKY Supertagger**).

Notice that the performance for the constituency metrics is not good even for the gold supertags case, but the dependency and SRL metrics are very good. As expected, the performance using the supertags predicted by the supertagger dramatically decreases, between 10 and 13 points for each metric. Improving the supertagger would yield better results for these metrics, but it still has the roof delineated by the gold supertags parser.

	U-Cons	L-Cons	U-Dep	L-Dep	U-SRL	L-SRL
CKY Gold tags	76.22	71.96	91.89	91.89	91.21	91.21
CKY Supertagger	65.83	59.26	83.03	80.58	83.73	78.28

Table 5.10: Performance for the CKY parsers over the development corpus. We show constituency F1, dependency accuracy and SRL F1 evaluations.

5.5 Top-down

The top-down parsing strategy takes a completely different approach. This strategy will try to leverage information from all the text sequence instead of the more localist approaches seen so far. In this case, the parsing process consists in a series of steps that incrementally build the parse tree: splitting a sentence into a binary tree, finding out the rules that apply to the nodes of the tree, and finally determining what nodes should be labeled with semantic argument categories.

5.5.1 Process

Let us walk through the proposed parsing process using again the sample sentence 39:

[El niño come una manzana roja]
(The kid eats a red apple)

Step 1: Splitter In the first stage, the process will take the whole sentence and split it in two sequences of words. The resulting subsequences are expected to be constituents of the sentence. In this case it should split the subject and the predicate. The result will look like the following:

[*El niño*] [*come una manzana roja*]

This process is repeated for each subsequence with three or more words, as sequences with two words are trivially split and sequences with only one word are already leaves in the tree. In this case, [*El niño*] has two words, so only the second subsequence will be split. The process should separate the verb from the object, resulting in the following:

[*come*] [*una manzana roja*]

Now the first subsequence has only one word (it is a leaf), and the process continues with the second subsequence separating the determiner from the rest of the noun phrase:

[*una*] [*manzana roja*]

At this point, there are no more non-trivial sequences to split, so the original sentences has been effectively transformed into a binary tree of words. The binary tree after step 1 for this example is shown in figure 5.14.

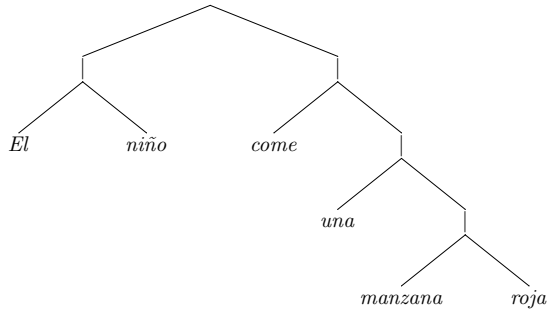


Figure 5.14: Tree for *"El niño come una manzana roja"* (*"The kid eats a red apple"*) after step 1.

Step 2: Rules After the tree is created, the second stage transverses all pairs of branches and tries to find the most suitable rule that describes the relation between those branches. For example:

[*El*] [*niño*] → `spec_head`

[*come*] [*una manzana roja*] → `head_comp`

[*manzana*] [*roja*] → `head_mod`

After this stage is completed, the tree looks as the one shown in figure 5.15.

Step 3: Arguments Once the tree is created and the syntactic features are in place, the third step tries to determine which of the syntactic arguments of the predicates (in our case all verbs, nouns and adjectives) should also be set as semantic arguments.

Given the rules determined after step 2 for each pair of branches in the tree, we can infer the head for each constituent. This third step considers each head and each target argument that belongs to that head. In our example:

[$\underline{\textit{El niño}}_{SPEC}$ $\underline{\textit{come}}_{HEAD}$ *una manzana roja*] → `arg0`

[*El niño* $\underline{\textit{come}}_{HEAD}$ $\underline{\textit{una manzana roja}}_{COMP}$] → `arg1`

[*una* $\underline{\textit{manzana}}_{HEAD}$ $\underline{\textit{roja}}_{MOD}$] → `none`

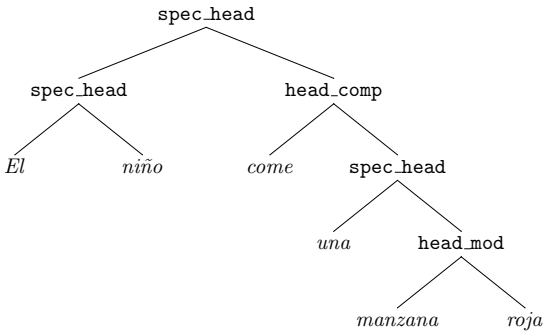


Figure 5.15: Tree for *"El niño come una manzana roja"* (*"The kid eats a red apple"*) after step 2.

A simplified version of the final tree for our example, after the three stages have been completed, is shown in figure 5.16. This final version of the tree contains the information about the arguments of each predicate and the semantic roles they have according to the argument structure.

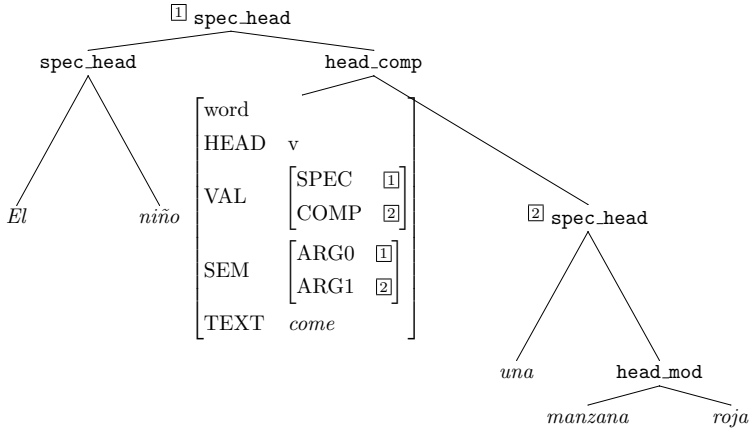


Figure 5.16: Simplified tree for *"El niño come una manzana roja"* (*"The kid eats a red apple"*) after step 3.

Notice that this top-down strategy does not use any subcategorization information for the lexical entries, for example it does not use the supertags like the CKY strategy does. Instead, this process just the uses the words of the sentence as input, encoded as word embeddings.

5.5.2 Neural network architectures

Each of the steps in this process can be implemented using a neural network. The architectures we chose for these networks in all cases include a central layer

with one or more bidirectional LSTMs. For our parsing process, we created the following models:

Step 1: Split model This model takes a sequence of words and returns a probability for each word. The word with the highest probability is the one the model considers the best split point for that sequence, i.e. the point in which the sequence could be separated in two constituents so that the process can be continued. The central part of this model is a three-layered stacked bidirectional LSTM network that returns the probability of each word in the sentence to be the split boundary.

This network was trained with sequences from the training corpus, so that the expected value for the correct split word was set to 1 and the rest of the words were set to 0.

The structure of the network is the following (see figure 5.17):

- Input: Sequence of words.
- Embeddings layer: Word embeddings of size 300.
- LSTM layers: Three layers of stacked bidirectional LSTMs.
- Dense layer: A fully connected layer.
- Output: Probability for each word.

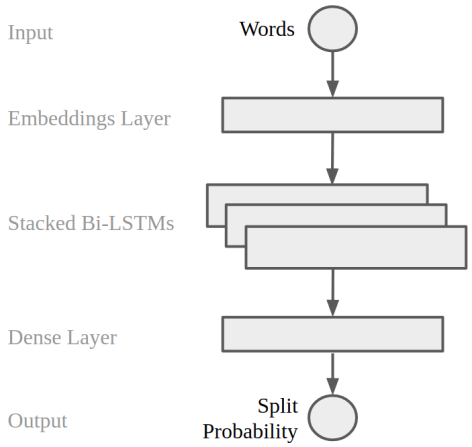


Figure 5.17: Architecture of the neural network for step 1.

Step 2: Rule model This model takes two constituents (two sequences of words) and returns the most likely rule that would define the relationship between them. The central part of this network are two parallel stacks of bidirectional LSTM layers that process the two sequences of words and return the

probability for each possible rule. One of the stacks is connected to the left constituent and the other one to the right one.

This network is trained using all pairs of subsequences that are subtrees of a tree in the training corpus and their corresponding rules. The two layers are updated independently during training.

The structure of the network (as shown in figure 5.18), is the following:

- Input: Two sequences of words.
- Embeddings layer: Word embeddings of size 300.
- LSTM layers: Two layers of parallel stacked bidirectional LSTMs.
- Dense layer: A fully connected layer.
- Output: Label indicating the rule to use (`head_spec`, `spec_head`, `head_comp`, `comp_head`, `head_comp_sem`, `head_mod`, `mod_head`, `clitic_head`, `head_rel`, `head_punct`, `punct_head`, `coord_left`, `coord_right`).

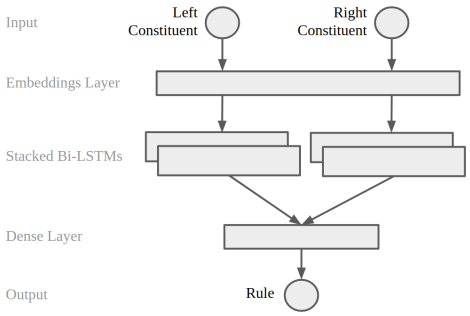


Figure 5.18: Architecture of the neural network for step 2.

Step 3: Argument model The network for this model is more complex than the other ones. In this case the network will take two sequences of words which are contained inside each other: one of them is the argument to classify, and the other is the largest constituent that contains the argument. It also takes the head of the constituent and the grammar rule that relates the head to the argument. The result of the network is the semantic role label that should be applied for this argument (or `none`).

The core of the model is a stack of three bidirectional LSTM layers, both input sequences (the argument and the other constituent) are run through the LSTM layers, and after that their result is concatenated together with the head and grammar rule information before calculating the final output.

This network is executed only for possible arguments of predicates, i.e. it is used for each verbal, nominal or adjectival head and all of their dependents. It is trained in the same way: using instances of <head, dependent> pairs from

the training corpus with verb, noun or adjective heads, whether their dependent is a semantic argument or not.

The structure of the network (as shown in figure 5.19), is the following:

- Input: Structured input with one word (the head, which is the predicate), two sequences of words (the whole constituent, and the target argument), and the syntactic valence of the target argument.
- Embeddings layer: Word embeddings of size 300. The embeddings layer is used for transforming the head and the two sequences of words.
- LSTM layers: Three layers of stacked bidirectional LSTMs.
- Dense layer: A fully connected layer. It takes as input the concatenation of the result of all the previous layers.
- Output: Label indicating the arg type (`none`, `arg0`, `arg1`, `arg2`, `arg3`, `arg4`, `argm`, `argl`, `argc`).

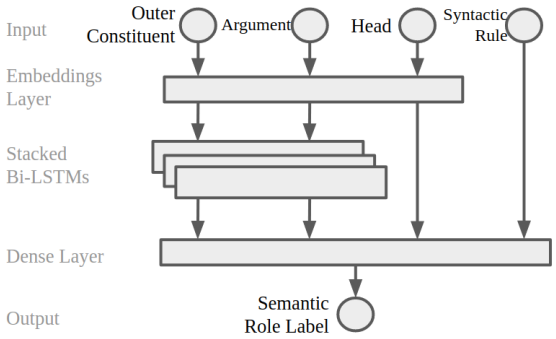


Figure 5.19: Architecture of the neural network for step 3.

Using these three networks we can implement the process as described so far following the three steps. However, as the three steps are performed independently, this presents two problems: the process is slow and the errors committed on an early step are carried along to other steps, which accumulates errors. Furthermore, as the steps are not trained at the same time, it cannot leverage information of one step to improve the others.

Because of this, in later experiments we created two alternative network architectures that attempt to merge the processing of different steps. One of the architectures merges steps 1 and 2 into a single network that is in charge of splitting a sequence and also determining the most likely rule to apply to the resulting segments. The other architecture merges the three steps into a single network that transverses the sequence and tries to predict all the information: where to split, which rule to apply, and if the corresponding dependent should be considered a semantic argument.

- **Split-Rule model** This model is similar to the step 1 split model, but instead of returning only the split probability for each word, it also returns the probability of rule application for each possible split point. The core of this model is a three-layered stacked LSTM network that transverses the sentence and returns, for each word, the probability of it to be the split boundary and the most likely rule to apply if that split point is used.

  The output of the network is in fact a `softmax` over all the possible rules plus the `none` rule for indicating that the word should not be used as a split point. After a sequence is processed, we first look at the word with the minimum score for the `none` output, which is selected as split point. Then we look at the corresponding rule probabilities for that point and select the most likely rule.

  This combined model has the advantage of being faster than using the models 1 and 2 independently, and it could also leverage the information used for both tasks (splitting and determining the rule) in order to improve the performance.

  The structure of the network is the following (see figure 5.20):

  - Input: Sequence of words.
  - Embeddings layer: Word embeddings of size 300.
  - LSTM layers: Three layers of stacked bidirectional LSTMs.
  - Dense layer: A fully connected layer.
  - Output: Split probability and rule to use for each word.

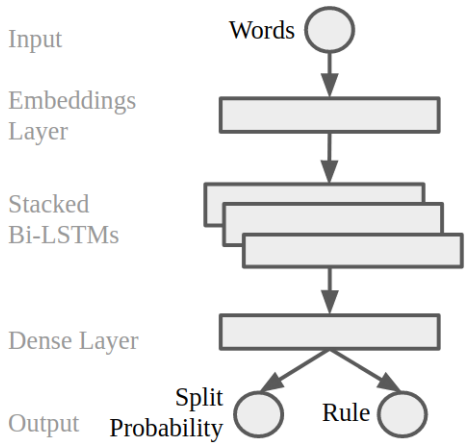


Figure 5.20: Architecture of the neural network for merged steps 1 and 2.

- **Split-Rule-Arg model** This model is also similar to the step 1 split model taking a sequence of words to split, but it returns all the information

necessary for the full parsing process: the most likely split point, and the grammar rule and semantic argument probabilities for each possible split point. As in the previous network, the central layer is a three-layered stacked LSTM network that transverses the sentence. The output is now divided in two: on the one side we have the split and rule scores as in the Split-Rule model, and on the other side we have the semantic role label to apply to this split (or the label `none` if it should not be considered a semantic argument).

This architecture has the advantage of being very fast, as it only has to be execute once for every subsequence, so it should run almost as fast as only performing the step 1 of the process. However, the information used by the argument detection step using the Arguments model considers more context and has more information than this Split-Rule-Arg model. We consider this is the reason why, as we will see in section 5.5.3, the experiments using this combined network achieve good results for splitting and detecting the grammar rule, but performs poorly for the semantic rule labeling step.

The structure of the network is the following (see figure 5.21):

- Input: Sequence of words.
- Embeddings layer: Word embeddings of size 300.
- LSTM layers: Three layers of stacked bidirectional LSTMs.
- Dense layer: A fully connected layer.
- Output: Split probability, rule, and argument label to use for each word.

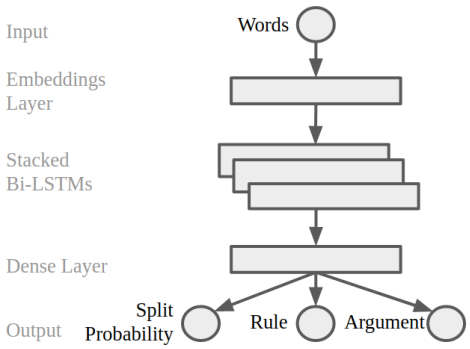


Figure 5.21: Architecture of the neural network for merged steps 1, 2 and 3.

Although we developed it independently, our LSTM top-down approach bears some similarities with [Stern et al., 2017], in that both approaches encode spans using a LSTM and use that information to find the best way to split a sequence of words. In our case the approach has the advantage that the

grammar is designed to be binary from scratch, so the strategy can leverage that in order to find better trees. Also, we add a SRL step, which was not one of the goals in [Stern et al., 2017].

5.5.3 Intrinsic evaluation by step

We will first present the intrinsic evaluation of the neural networks that implement the different steps of the process described so far. The intrinsic evaluation for each step involves evaluating how well the step works in isolation using a model. This could mean different things for the different steps. The evaluation metrics we use in this section are the following:

- **Split accuracy:** Given a sequence, how often the model indicates the correct split word position with a higher probability than the rest. For this evaluation we will only present the model correct sequences that would appear during the parsing history of the development set, so this intrinsic evaluation will be artificially high. In a real scenario, mistakes at the beginning of the parsing process would pile up making further mistakes more likely.
- **Rule accuracy:** How often the model selects the correct rule to apply when presented with a pair of sequences of words that represent two correct subtrees. As in the split case, we will only present the model correct pairs of sequences that would appear on the development set, so the evaluation will be artificially high with respect to a real scenario.
- **Args accuracy:** How often the model predicts the correct semantic role label for a sequence of words with respect to its predicate. The model will only be evaluated with correct constituents and predicates (verbs, nouns, and adjectives) that appear in the development corpus.
- **Split-Rule accuracy:** For combined models that are able to predict split and rule information at the same time, we will also evaluate the how often the model indicates the correct split point and also the correct rule is the one with the highest probability at that point.

Using the different models described, there are three possible configurations we can define for combining the models in order to execute the full process:

- **Independent models:** Using one neural network for each one of the steps, i.e. using the separate Split, Rule and Argument models.
- **Split-Rule combined + Arg model:** Using a combined neural network for performing steps 1 and 2, and another network for step 3.
- **Split-Rule-Arg combined model:** Using only one neural network for performing the three steps.

	Split Acc.	Rule Acc.	Split-Rule Acc.	Args Acc.
Split Model	89.99	-	-	-
Rule Model	-	98.11	-	-
Argument Model	-	-	-	93.62
Split-Rule Model	94.38	96.22	92.81	-
Split-Rule-Arg Model	93.37	95.70	91.95	87.83

Table 5.11: Intrinsic accuracy for the split, rule and argument steps using the different independent and merged models.

The objective of the intrinsic evaluation is to analyze which of the three configurations would yield the best results so we can focus only on that configuration for the rest of the work and discard the others.

Table 5.11 shows the intrinsic accuracy for each step and for each one of the models described above. In the table, the models that perform only one step only have the accuracy for that step. As can be seen in the table, the Split Model has the lowest intrinsic accuracy for step 1 of all the models. Even if the independent Rule Model seems to be the best one for calculating the rule in terms of intrinsic accuracy, if we use it after applying the Split Model, its combined split-rule accuracy would never be more than the split accuracy achieved by the Split Model.

On the other hand, the combined Split-Rule and Split-Rule-Arg models achieve high split accuracy and almost as good rule accuracy, which translates into a high combined split-rule accuracy. The Split-Rule Model seems to have a small advantage over the Split-Rule-Arg Model. Consequently, if we consider that the arguments accuracy for the independent Argument Model is considerably higher than the accuracy achieved by the Split-Rule-Arg Model, it seems reasonable that the best configuration to use in terms of accuracy would be **Split-Rule combined + Arg model**, which leverages the strengths of combining the split and rule steps, and the good accuracy of the independent arguments model.

5.5.4 Structural error analysis

As mentioned in section 5.1.2, this approach that starts by predicting the tree and deriving the lexical entries from it does not have the limitation of the CKY with supertagging approach in that classes of lexical entries not seen in the training corpus could not be used. On the contrary, this approach is free to generate lexical entries with any kind of features it needs in order to accommodate the predicted tree. However, this comes with a caveat: it is possible that the generated trees violate some of the constraints imposed by our grammar or our feature structures.

We ran a set of experiments on the results of the corpus for the development set to try to see to what extent the resulting trees are well formed. Although our grammar does not impose many constraints on how the trees are built, there are

still some structural properties that should be held. We focused on the following aspects:

- Invalid coordinations: Cases where there was a `coord_left` without being immediately followed by a `coord_right` in the tree, or cases where there was a `coord_right` and the parent structure is not `coord_left`.
- Invalid relatives: Cases where there is a `head_rel` construction but the structure on the right does not start with a relative pronominal expression.
- Double specifiers: There should be at most one specifier per head, so we counted the times a lexical entry ended up with more than one specifier attached.
- Invalid clitics: How many times the `clitic_head` rule was applied but the left element was not a clitic pronoun.
- Triple clitics: In Spanish there can be up to two clitic pronouns accompanying a verb, so we count how many times there system attached three or more clitics to a verb.
- Specifiers on the right of nouns: The `head_spec` rule was designed with the case of postponed subjects in mind, but as we use the same specifier rules for modeling a determiner-noun construction, there is a possibility that the system erroneously attaches a determiner on the right of a noun as a specifier.

Error type	Occurrences
Invalid coordinations	107
Invalid relatives	90
Double specs	104
Invalid clitics	3
Triple clitics	0
Right noun specifiers	0

Table 5.12: Number of occurrences over the development corpus for the different structural errors.

Table 5.12 shows the number of times these errors occur in the results of the parser for the development corpus. We can see that there are indeed some cases in which the method incurs in some of these structural errors, particularly for the first three structural error types. However, the number of structural errors committed is rather low: only around 0.5% of the predicted structures contain some error. Notice that this is a internal structural evaluation without comparing to the gold trees, but any of these errors will necessarily imply a deviation from the gold tree, so these kinds of errors are already contemplated in the performance metrics as well. We consider that, as the proportion of errors is low, this will not make much of a difference when comparing the performance against other parsers.

5.6 Training details

The following applies to all the neural network models we trained. The models were implemented in the `keras` library [Chollet, 2015] over `tensorflow` [Abadi et al., 2015]. The word embeddings layer uses the collection described in section 5.3. It is an embeddings collection of 1,146,242 vectors of dimension 300, and we created an unknown token for each POS tag (considering morphological information) for handling out of vocabulary tokens.

We tuned several hyperparameters against the development corpus, but we are only reporting the results of the best performing model for each set of experiments. We applied dropout to LSTM and dense layers, in general varying between 0.1 and 0.5. We trained several configurations for every experiment varying the number of units (between 100 and 600 units) and the activation functions (generally `relu` or `tanh`) for each layer. When using a stack of LSTM layers, we varied between one and three layers. For the bottom-up baseline MLP architectures, we varied between one and two dense layers.

We also used the early stopping technique during training, with a held-out set of between 10% and 20% of each training set, depending on the experiment.

5.7 Evaluation of approaches

Table 5.13 shows a summary of the results over the development corpus for the approaches described in this chapter, including the LSTM Top-down architecture that uses the Split-Rule combined + Arg model configuration. The results for the CKY approach with gold supertags is shown for completeness, as it is an unrealistic approach.

From the table we can conclude that the best approach of the ones described is the LSTM top-down approach. Its performance is better than the other approaches on all metrics, and it is even better than the CKY with gold supertags method for the constituency metrics. Nonetheless, there seems to be a wide gap between the unlabeled and labeled metrics for this approach, both in terms of constituency, dependency and SRL metrics. The gap for the top-down model looks wider than the gap for the CKY approach. This might be explained by the fact that the CKY encodes information about head-dependency, grammar rule and semantic role label at the same time using the supertags, so once the supertag is correctly predicted, guessing the appropriate relation between a head and a dependent implies guessing the correct rule, and also the correct semantic role. On the other hand, the top-down approach is more prone to select a correct partition between a head and a dependent, but it fails to label the relationship appropriately for the grammar rules or for the semantic roles.

Approach	U-Cons	L-Cons	U-Dep	L-Dep	U-SRL	L-SRL
Bottom-up Bilex Context 0	49.17	38.40	63.37	59.92	59.14	45.72
Bottom-up Bilex Context 1	61.13	47.17	65.36	59.81	58.93	44.58
Bottom-up MLP Context 1	70.53	61.46	79.96	75.17	65.56	49.21
Bottom-up MLP Context 2	74.18	65.17	81.57	76.70	67.35	50.41
CKY Gold tags	*76.22*	*71.96*	*91.89*	*91.89*	*91.21*	*91.21*
CKY Supertagger	65.83	59.26	83.03	80.58	83.73	78.28
LSTM Top-down	**87.17**	**81.71**	**90.92**	**88.60**	**87.40**	**80.23**

Table 5.13: Performance for all the approaches over the development corpus. We show constituency F1, dependency accuracy and SRL F1 evaluations.

Chapter 6

Evaluation

So far we have presented a comparison between our parsing strategies evaluated over the development corpus, which was used for parameter tuning. In this chapter we show the experimental results for our parsing strategies over the test corpus and compare them with other Spanish parsers in terms of syntactic parsing and semantic role labeling. A preliminary version of this comparison is presented in [Chiruzzo and Wonsever, 2020]. We will also make a comparison of execution times and an analysis of the performance for some particular phenomena.

6.1 Evaluated systems

We compare the strategies defined in chapter 5 with other six well established baselines for Spanish parsing. Unfortunately, none of the external baselines works with a formalism similar to the one we use, so we will only be able to compare some of the metrics for them. The only other parser that would use a similar structure that we know of (the Spanish Resource Grammar) is not available for comparison. It also has problems for parsing longer sentences, and there are many of these in the corpus.

- Bottom-up baselines: These are the approaches presented in section 5.3, both using simple counts of words or using multi-layer perceptrons for considering more context. These are the systems **Bottom-up Bilex Context 0**, **Bottom-up Bilex Context 1**, **Bottom-up MLP Context 1** and **Bottom-up MLP Context 2**.

- CKY approaches: We consider two approaches based on CKY, as presented in section 5.4. One of them is the unrealistic model that starts from the gold supertags, which we consider is an upper bound for the CKY process (**CKY Gold tags**). The other one is the more realistic CKY process that first uses a supertagger to predict the lexical frames it

will have to use, and it will forcefully perform worse than the previous one (system **CKY Supertagger**).

- LSTM top-down system: This is the approach described in section 5.5, which uses a greedy top-down process with a LSTM for finding the structure and rules, and a second neural network for calculating the semantic arguments (system **LSTM Top-down**).

- FreeLing: FreeLing [Padró and Stanilovsky, 2012] is an important suite of NLP tools developed by Universitat Politècnica de Catalunya that has several Spanish models which include both dependency parsing and SRL. The ones we used for this comparison are:

  - **FreeLing Txala** [Batalla et al., 2005; Lloberes et al., 2010] is a dependency parser based on transforming the output of a previous rule-based parser into dependency format.
  - **FreeLing Treeler**[1] is a parser that uses a factorization or decomposition model trained using several statistical models (log linear, max-margin and perceptron).
  - **FreeLing LSTM** is a LSTM implementation of a dependency parser trained over AnCora.

- spaCy: spaCy[2] is a suite of NLP tools developed by Explosion AI. It contains models for several languages and in particular two dependency parsers and named entity recognizers for Spanish. These models are convolutional neural networks trained with multi-task training over the Universal Dependencies conversion of AnCora and the WikiNER corpus. We use models `es_core_news_sm` (small) and `es_core_news_md` (medium), the large model was not available. These models perform dependency parsing but not SRL.

- UDPipe: UDPipe [Straka and Straková, 2017] is a trainable pipeline that provides tokenization, morphological analysis and dependency parsing using the Universal Dependencies format. It was used as base implementation in CoNLL competitions. The parser uses a transition-based algorithm implemented with a simple neural network with one hidden layer, it is very fast and robust. They provide pre-trained models for several languages. We are using model `Spanish-AnCora`, which was trained with the Universal Dependencies conversion of AnCora. Like the spaCy models, this parser returns dependency parsing but not SRL.

[1] `http://treeler.lsi.upc.edu/`
[2] `https://spacy.io`

6.2 Global metrics

The global metrics we used are the ones defined in section 5.2: Unlabeled Constituency F1 (**U-Cons**), Labeled Constituency F1 **L-Cons**, Unlabeled Dependency Accuracy (**L-Dep**), Labeled Dependency Accuracy (**U-Dep**), Unlabeled SRL F1 (**U-SRL**) and Labeled SRL F1 (**L-SRL**). Notice, however, that not all metrics are applicable directly to all parsers.

First of all, the constituency metrics only apply to our parsers, as we are using a rather different grammar formalism. Our constituents are binarized and are labeled with the rule that was applied to form them. This is not the case for the only other Spanish constituency parser we could use, which belongs to the FreeLing suite. In this case, the constituents are not binarized and are annotated with a wide range of categories. The constituents in these trees tend to be very flat, not unlike AnCora. Given the complexity that a conversion between our format and FreeLing's constituency format would have, we decided to try a simpler approach that is doing the comparison only for dependencies, as FreeLing allows dependency outputs for all its parsers.

The dependencies comparison is not exempt from problems either. As mentioned in section 2.1.3, different dependencies formalisms might differ in what elements they consider should be the heads of structures. For example, when transforming a prepositional phrase to dependencies, in our case we consider that the preposition is the head of the structure. The parsers in the FreeLing suite work in the same way. However, parsers based on Universal Dependencies like spaCy or UDPipe give precedence to the content words over the function words to assign them as heads, so in those cases the preposition would be a dependent instead of a head. In order to compare to these parsers, we post-processed the results and transformed the heads of prepositional phrases, copulas and other structures in order to adapt it to our format.

On the other hand, the labels we use in our dependency conversion are the grammar rules used to attach the dependant at each step, so they are a completely different label set from the ones used by the other parsers. The different parsers also use different dependency tag sets, even the parsers in the FreeLing suite have a different tagset (Treeler and LSTM use one tagset but Txala uses a different one), while spaCy and UDPipe use the same tagset defined by Universal Dependencies. Given this disparity of scenarios, we decided to only perform the full comparison with all the metrics for our parsing strategies, but only use the U-Dep metric when comparing to external parsers. The U-Dep metric is the most similar to standard Unlabeled Attachment Score, with the caveat that we are transforming the output of some parsers to comply to our definition of dependencies.

Lastly, the semantic role labeling metrics should be comparable across parsers because we defined them over them as dependencies between the heads of the constituents. Unfortunately, not all the parsers provide the SRL information, and in this case we can only compare to the parsers in the FreeLing suite.

6.2.1 Syntactic parsing

The performance results over the test corpus for the different systems are shown in table 6.1. The best performing model according to this table is the LSTM top-down approach, both for constituency and dependency metrics. It is rather surprising that the CKY method underperformed for the constituency metrics, even when starting with the gold supertags which is the upper bound for the CKY methods. We can see that the unlabeled constituency metric gets about 77% for this method, while the corresponding dependency metric jumps to 92%. This might indicate that the main issue is that the method is able to identify the heads and dependents correctly, but misses the order of application of the rules frequently. The rather weak statistical model, than only takes in consideration partial information from the supertags, might be one reason why these models are underperforming. In the future we might try to enrich the statistical model using more fine-grained information like the lexical entries being used or the partial derivation history when applying the rules.

The LSTM top-down approach is very robust, outperforming all our other strategies and even getting results for dependency that are almost as good as with the CKY using gold tags. As expected, the bottom-up processes and the CKY using the supertagger results have much lower performance for all the metrics.

Compared to the external baselines, the results seem to indicate that our top-down parser also outperforms them for this corpus. However, we must take in consideration that the post-processing probably added some noise over the comparison, because the structures are not exactly the same as the original and also because we cannot be sure that the conversion of all structures that behave differently was done exhaustively.

Model	U-Cons	L-Cons	U-Dep	L-Dep
Bottom-up Bilex Context 0	49.39	38.51	63.49	60.10
Bottom-up Bilex Context 1	60.97	47.45	65.43	60.05
Bottom-up MLP Context 1	70.83	61.52	79.83	75.12
Bottom-up MLP Context 2	74.38	65.42	81.59	76.77
CKY Gold tags	*77.16*	*72.72*	*92.07*	*92.05*
CKY Supertagger	66.08	59.33	83.34	81.03
LSTM Top-down	**87.57**	**82.06**	**91.32**	**88.96**
FreeLing LSTM	-	-	83.15	-
FreeLing Treeler	-	-	83.61	-
FreeLing Txala	-	-	69.75	-
spaCy es_sm	-	-	83.01	-
spaCy es_md	-	-	83.69	-
UDPipe	-	-	82.09	-

Table 6.1: Results of the syntactic parsing experiments over the test set.

6.2.2 Semantic Role Labeling

The results for Semantic Role Labeling are shown in table 6.2. The upper bound in this case is given by the CKY using gold tags, getting around 88% for the SRL F1 metric, and a slight difference of one point between the unlabeled and labeled versions. It is expected that the unlabeled and labeled metrics would behave similarly for this case, because we are sure the parser starts with the appropriate supertags, so if it can predict the head-dependent pair correctly, it will certainly predict the appropriate semantic role for it as it is encoded in the supertag. The CKY using supertags, however, performs worse for detecting the head-dependent pair, and even worse for assigning the appropriate semantic role. This might be explained in part by the fact we observed empirically that the supertagger tends to predicts supertags without semantic role features more likely than the versions with semantic roles.

The LSTM top-down approach gets almost as good results for the unlabeled metric as the CKY with gold tags. However, it gets behind for the labeled metric, which indicates is frequently predicts a syntactic argument as semantic argument, but fails to assign the appropriate label.

Both CKY and top-down approaches outperform all the external baselines based on FreeLing, which behave more or less as our bottom-up baselines.

Model	U-SRL	L-SRL
Bottom-up Bilex Context 0	59.73	45.82
Bottom-up Bilex Context 1	60.46	45.63
Bottom-up MLP Context 1	66.61	49.90
Bottom-up MLP Context 2	68.21	50.71
CKY Gold tags	*88.51*	*87.51*
CKY Supertagger	81.48	75.78
LSTM Top-down	**87.68**	**80.66**
FreeLing LSTM	68.50	60.74
FreeLing Treeler	69.10	61.53
FreeLing Txala	52.17	45.73

Table 6.2: Results of the semantic role labeling experiments over the test set.

6.2.3 Execution time

Besides the global performance metrics, we compared the different parsers in terms of their speed based on the global execution time over our test corpus. This comparison is more related to the different type of algorithms used by each parser and might be skewed by how much fine-tuned or optimized they are. Table 6.3 shows the average time for parsing a sentence in the test set for the different models. The experiments were run on an Intel i7, 2.7GHz, 16GB RAM, without GPU acceleration. The metrics in the table are an average over all sentence lengths, with 1,692 sentences in total.

Model	Time (ms)
spaCy es_sm *(no SRL)*	9.3
Bottom-up Bilex Context 0	18.8
spaCy es_md *(no SRL)*	19.8
UDPipe *(no SRL)*	21.8
FreeLing Txala	41.8
FreeLing LSTM	60.3
Bottom-up MLP Context 1	63.9
Bottom-up MLP Context 2	78.0
LSTM Top-down	86.1
Bottom-up Bilex Context 1	88.3
CKY Gold tags	288.7
FreeLing Treeler	948.5
CKY Supertagger	1,237.3

Table 6.3: Average time in milliseconds for parsing a sentence in the test set.

There seems to be an advantage in terms of speed for the parsers based on neural networks over the parsers based on other (statistical) techniques, except our simplest bottom-up baseline, which is very fast (but, as we have seen, performs very poorly for the global metrics). The fastest parsers are spaCy and UDPipe (besides the simplest baseline), but we have to consider that these parsers only perform the syntactic dependency analysis, and they do not provide SRL information. This could be one of the reasons they are so fast. Around the middle of the table (between 40 ms and 90 ms per sentence) we can see the most of the parsers, including the LSTM top-down parser which is only slightly faster than the slower baseline. On the trailing positions we find the CKY parsers and FreeLing Treeler. The CKY with supertagger parser is the worst of all, taking more than a second for each sentence on average, which can be explained by the high number of times the back-off rules have to be enabled due to invalid combinations of supertags returned by the supertagger (around one word out of ten will have an incorrect supertag).

If we break down the execution time of the LSTM top-down process, we get that there is a balance in the time spent at each step: 54.1 ms for syntactic parsing and 31.9 ms for argument identification. This could be sped up if we used a unified architecture that could handle the splitting, rule prediction and semantic arguments identification steps at the same time. However, as seen in section 5.5, this architecture underperformed for other metrics, so we kept two separate networks.

The times shown in table 6.3 are average over all sentences in the test corpus, but given the differences between parsing algorithms, it would be interesting to see how well they behave for different sentence lengths. Figure 6.1 shows a breakdown of execution times for our approaches when parsing sentences of different length, up to 80 words long. In this case we show only the Bottom-up MLP2 baseline because it is the best performing of the simple baselines.

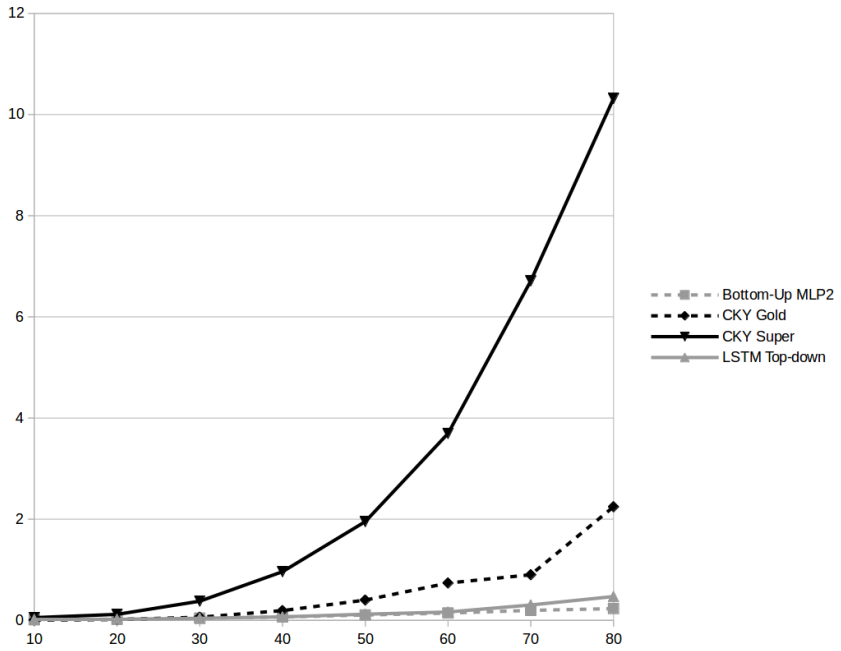


Figure 6.1: Breakdown of execution time in seconds for different input sizes.

The LSTM top-down and the MLP Bottom-up execution times seem to grow close to linearly while for the CKY approaches it grows faster. One explanation for this might be that the growth rate for both LSTM top-down and MLP bottom-up (quasi-linear or quadratic) is lower than the one for CKY (at least cubic). Particularly the CKY with supertagger grows much faster than the others, which can be explained because, especially for longer sentences, the probability of obtaining a sequence of tags that does not form a correct tree is much higher as the sentence grows, so the fallback rules have to be enabled more frequently, rendering the process much slower.

6.3 Particular phenomena

We described in chapter 3 the grammar we are using and mentioned some of the Spanish phenomena we are trying to model, which in some cases led us to make particular choices for designing the rules and features. We now want to analyze to what extent our parsers are able to capture these phenomena in actual sentences. In this section we will define a set of metrics over some particular Spanish phenomena, which cover some aspects of the language we wanted to address. As was the case with the dependency results conversion, in these cases it was also necessary to post-process the results of the external parsers so as to adapt the different ways they represent these phenomena. Once again, this post-processing might add some noise in the results, so the comparisons shown in this section might hint at some differences between parsers, but further research would be needed to confirm these results.

6.3.1 Postponed subjects

As seen in section 3.3, although in Spanish the usual position for a subject is on the left of the verb (SVO style), it is also very common for some verbs to write the subject on the right. We wanted to know how much the parsers were able to identify a subject that occurs on the right of the verb. There are 542 instances of postponed subjects in the test corpus. Table 6.4 shows the performance in terms of precision, recall and F1 score of all the systems for this metric.

	Precision	Recall	F1 Score
Baseline Bilex Context 0	28.45	12.91	17.76
Baseline Bilex Context 1	52.87	8.48	14.60
Baseline MLP Context 1	80.45	19.74	31.70
Baseline MLP Context 2	68.69	31.18	42.89
CKY Gold tags	*99.25*	*98.89*	*99.07*
CKY Supertagger	**84.25**	**76.01**	**79.92**
LSTM Top-down	82.22	64.02	71.99
Freeling Txala	60.00	19.92	29.91
Freeling LSTM	69.74	65.49	67.55
Freeling Treeler	74.07	62.73	67.93
spaCy es_sm	60.11	56.45	58.23
spaCy es_md	59.92	59.59	59.75
UDPipe	57.37	52.39	54.77

Table 6.4: Results of the parsers for postponed subject identification.

We can see that CKY with supertagger performs better for this case, mainly because it has a better recall than LSTM top-down and their precisions are similar. When analyzing the errors, we found out that LSTM top-down generally attaches correctly the head to the subject, but most of the time it mistakenly applies a `head_comp` rule instead of a `head_spec` rule.

One example of this is the following, which involves a relative clause:

- *"la reunión que mantendrá$_H$ este jueves [$_S$ el Consejo_de_Administración del club]"*

  "the meeting that the Administrative Council of the club will hold on Thursday"

This might indicate that the reason these cases are difficult to classify is that the context that says the overall shape of the structure, in this case a relative construction, is not available for the top-down parser in the moment it is trying to define the relation between the head and the subject candidate. This is better resolved by the CKY parser because the supertagger can leverage the full context of the sentence when predicting the supertag for the verb. If the supertag is predicted correctly, it will indicate that a postponed subject is expected.

6.3.2 Clitics

The use of clitic pronouns in Spanish motivated us to create a special rule to handle them (see section 3.7). We wanted to know if our parsing strategies are any good at associating the clitic pronouns to their corresponding verbs (which should be an easy task) and also if they are able to detect the semantic role they correspond to. Table 6.5 shows the performance for these two experiments: clitics identificacion and classification. As classification is a multi-class problem, we show the accuracy and macro-averaged metrics.

	Identification			Classification			
	Prec	Rec	F1	Acc	M-Prec	M-Rec	M-F1
Bottom-up Bilex Context 0	75.19	80.46	77.74	56.40	17.52	20.38	18.84
Bottom-up Bilex Context 1	80.00	78.67	79.33	57.63	18.31	20.83	19.49
Bottom-up MLP Context 1	96.38	91.60	93.93	66.57	18.17	24.06	20.70
Bottom-up MLP Context 2	94.23	92.15	93.18	66.16	17.95	23.91	20.50
CKY Gold tags	*100.*	*100.*	*100.*	*95.46*	*98.46*	*88.18*	*92.78*
CKY Supertagger	98.71	95.46	97.06	83.63	65.05	57.80	61.00
LSTM Top-down	**98.76**	**99.03**	**98.90**	**85.28**	**76.20**	**66.52**	**70.60**
Freeling Txala	96.44	85.83	90.82	75.10	72.03	44.02	49.20
Freeling LSTM	78.25	88.58	83.09	80.47	82.67	51.51	56.37
Freeling Treeler	80.67	88.44	84.38	80.88	83.50	54.60	58.63
Spacy-sm	84.99	97.38	90.76	-	-	-	-
Spacy-md	83.47	97.93	90.12	-	-	-	-
UDPipe	82.33	96.83	89.00	-	-	-	-

Table 6.5: Results of the parsers for clitics identification and classification.

A special phenomenon we are interested in analyzing is the detection of clitics reduplication. In order to do this, we focused on detecting the cases where there is a clitic and another argument of the verb (subject or complement) that share the same semantic role value, so this could only be calculated for the parsers that return semantic role labels. There are 48 instances of clitics reduplication in the test corpus. Table 6.6 shows the results of the clitics reduplication comparison.

	Precision	Recall	F1 Score
Bottom-up Bilex Context 0	0.00	0.00	0.00
Bottom-up Bilex Context 1	0.00	0.00	0.00
Bottom-up MLP Context 1	0.00	0.00	0.00
Bottom-up MLP Context 2	0.00	0.00	0.00
CKY Gold tags	*100.*	*81.25*	*89.65*
CKY Supertagger	**75.00**	18.75	30.00
LSTM Top-down	32.69	**35.41**	**34.00**
Freeling Txala	8.33	2.08	3.33
Freeling LSTM	22.05	31.25	25.86
Freeling Treeler	30.23	27.08	28.57

Table 6.6: Results of the parsers for clitics reduplication identification.

Detecting clitics reduplication seems to be a hard problem for all the parsers. In this case the CKY with supertagger approach has much higher precision but very low recall. It might be difficult to solve this by the supertagger because, as mentioned before, it has a tendency to assign supertags that do not indicate a semantic argument, instead of assigning the ones that do. For detecting clitics reduplication, we need a supertag that has at least two semantic argument markers. On the other hand, LSTM top-down is much more consistent between precision and recall, which also seems to be the case for the external parsers.

6.3.3 Relative clauses

The only type of long range dependency we are modeling is the use of relative clauses as modifiers of nouns, as described in section 3.8. It is interesting to see if our parsing strategies are good at handling these scenarios. First of all, the relative clauses identification metric focuses on measuring if the parsers are able to detect a relative pronominal expression that is attached to a subordinate sentence in order to create a relative clause. We also want to know if the corresponding relative pronominal expression is classified correctly with respect to SRL. For example, in the phrase *"el perro que Juan vio"* (*"the dog that John saw"*), we would like the parsers to detect that the pronoun *"que"* is attached to *"vio"* forming a relative clause, and also acts as `ARG1` of the verb. Table 6.7 shows the results of these metrics.

Secondly, we focused on measuring how well the parsers could correctly attach a relative pronominal expression both to its verb and the corresponding nominal referent the expression points to. In the phrase *"el perro que Juan vio"*, it should identify that the pronoun *"que"* is attached to the verb *"vio"* and also to the noun *"perro"*. There are 737 cases of this in the test corpus. Table 6.8 shows the results for this metric.

	Identification			Classification			
	Prec	Rec	F1	Acc	M-Prec	M-Rec	M-F1
Bottom-up Bilex Context 0	22.05	4.07	6.87	2.04	20.09	0.90	1.72
Bottom-up Bilex Context 1	29.85	8.14	12.79	3.39	14.11	1.60	2.79
Bottom-up MLP Context 1	44.88	41.65	43.20	16.96	10.69	8.41	8.37
Bottom-up MLP Context 2	48.11	45.04	46.53	16.69	10.72	8.17	8.34
CKY Gold tags	*100.*	*99.72*	*99.86*	*81.14*	*84.36*	*78.47*	*72.39*
CKY Supertagger	**92.27**	81.00	86.27	57.67	54.32	37.82	43.29
LSTM Top-down	90.60	**89.00**	**89.80**	**76.12**	**64.73**	**55.32**	**59.02**
Freeling Txala	72.84	54.95	62.64	4.21	30.54	7.63	6.54
Freeling LSTM	76.59	58.61	66.41	43.15	40.75	24.78	29.18
Freeling Treeler	78.23	59.02	67.28	42.06	40.76	24.39	28.62
Spacy-sm	69.16	55.08	61.32	-	-	-	-
Spacy-md	70.85	55.35	62.05	-	-	-	-
UDPipe	66.49	52.23	58.51	-	-	-	-

Table 6.7: Results of the parsers for relative clauses identification and classification.

	Precision	Recall	F1 Score
Bottom-up Bilex Context 0	16.91	3.12	5.26
Bottom-up Bilex Context 1	24.87	6.78	10.66
Bottom-up MLP Context 1	37.28	34.59	35.89
Bottom-up MLP Context 2	38.55	36.09	37.28
CKY Gold tags	*77.68*	*77.47*	*77.58*
CKY Supertagger	67.69	59.43	63.29
LSTM Top-down	**73.61**	**72.32**	**72.96**
Freeling Txala	37.23	28.08	32.01
Freeling LSTM	60.46	46.26	52.42
Freeling Treeler	63.30	47.76	54.44
Spacy-sm	54.17	43.14	48.03
Spacy-md	55.53	43.55	48.82
UDPipe	49.56	38.94	43.61

Table 6.8: Results of the parsers for relative referents identification.

Analyzing the cases LSTM top-down got right but CKY got wrong, we noticed that it is usual that CKY has mistakes like the examples shown in figure 6.2. In both cases there is a mismatch between the numbers of the verb and the noun selected by the parser, while the correct noun should agree with the verb. This could happen because the statistical model used by the CKY algorithm is too simple, it does not use any lexical information so it does not know whether the verbs and nouns are plural or singular. On the other hand, LSTM top-down uses lexical information in the form of word embeddings for all its decisions, so it should be better suited to handle these situations. This might explain in part the better performance obtained by the LSTM top-down parser in this case.

- *"un anticipo$_{GOLD}$" mínimo de dos_por_ciento " de sus ingresos$_{CANDIDATE}$ brutos , que$_{REL}$ sería$_{VERB}$ deducible de (...)"*

  "a minimum advance of two percent of their gross income, which would be deductible from (...) "

- *"un proyecto$_{CANDIDATE}$ de prevención del daño y atención sanitaria , que además persigue " devolver la dignidad " a personas$_{GOLD}$ que$_{REL}$ viven$_{VERB}$ en (...)"*

  "a harm prevention and health care project that also aims to "restore dignity" to people living in (...) "

Figure 6.2: Two examples of relative referents that LSTM top-down hit but CKY with supertagger missed. We show the expected relative referents ($_{GOLD}$), the referents that CKY chose ($_{CANDIDATE}$), and also the corresponding relative pronouns ($_{REL}$) and the verbs they are attached to ($_{VERB}$).

6.3.4 Coordination chains

As seen in section 3.10, in our grammar we model chains of coordinations using the `coord_left` and `coord_right` rules. We wanted to see how well the different parsers capture the chains of two or more coordinated elements present in the corpus. We are trying to find chains of elements like *"peras, manzanas y naranjas"* and we expect the parsers to recognize the exact whole chain. There are 1432 chains of coordinated elements in the test corpus. Table 6.9 shows the results for this experiment.

	Precision	Recall	F1 Score
Bottom-up Bilex Context 0	8.85	19.97	12.26
Bottom-up Bilex Context 1	16.33	5.09	7.77
Bottom-up MLP Context 1	30.00	21.57	25.10
Bottom-up MLP Context 2	30.57	24.79	27.38
CKY Gold tags	*74.02*	*77.23*	*75.59*
CKY Supertagger	54.92	48.25	51.37
LSTM Top-down	**65.49**	**65.22**	**65.36**
Freeling Txala	24.48	22.20	23.28
Freeling LSTM	56.53	53.49	54.96
Freeling Treeler	56.09	53.00	54.50
Spacy-sm	41.39	43.85	42.59
Spacy-md	42.47	44.34	43.38
UDPipe	37.22	39.59	38.37

Table 6.9: Results of the parsers for coordination chains identification.

6.3.5 Verbs analysis

Finally, we wanted to analyze some aspects related to the modeling of verbs in the corpus and in our grammar. These metrics only make sense for our parsers, because we focused on modeling AnCora's verbal periphrases (like *"estaba empezando a cantar"*) in a special way using the `head_comp_sem` rule (see section 3.5). There are 826 instances of verb periphrases in the test corpus. Table 6.10 shows the result of the comparison for our parsing strategies.

	Precision	Recall	F1 Score
Bottom-up Bilex Context 0	0.00	0.00	0.00
Bottom-up Bilex Context 1	0.00	0.00	0.00
Bottom-up MLP Context 1	83.91	70.09	76.38
Bottom-up MLP Context 2	86.52	66.10	74.94
CKY Gold tags	*96.92*	*87.77*	*92.12*
CKY Supertagger	82.75	69.73	75.68
LSTM Top-down	**92.47**	**93.70**	**93.08**

Table 6.10: Results of the parsers for verb periphrases identification.

LSTM top-down clearly outperforms the other strategies. The main difference in performance between the CKY and the LSTM top-down approaches is in the recall. Manually inspecting the errors commited by the CKY parser, we could see that most of the times it makes a mistake, there are one or more `head_none` or `none_head` applications in the expected verb periphrasis chain. As verb periphrases are generally comprised of words that are together in the text, this might indicate that the main source of these errors is the supertagger, which might be returning an incompatible sequence of tags, and not the probabilistic model.

Another aspect we wanted to analyze was the capacity of the parsers to detect impersonal verbs, i.e. verb instances that do not have any subject. In our parsers we distinguish this from null subjects because we use AnCora's null subject markup, which is not possible for the other parsers, so this metric is only calculated for our parsing strategies. There are 1051 instances of this kind of verbs in the test corpus. Table 6.11 shows the results of this comparison for our parsers.

In this case, the recall is generally good for all approaches. CKY with supertagger has particularly low precision, which might indicate that it is skipping an important number of subjects, so it mistakenly predicts that many of those verbs are impersonal. LSTM top-down is the one with best overall performance because it is the one with the highest precision in this case.

	Precision	Recall	F1 Score
Bottom-up Bilex Context 0	27.11	94.38	42.13
Bottom-up Bilex Context 1	31.21	95.05	47.00
Bottom-up MLP Context 1	48.76	**96.19**	64.72
Bottom-up MLP Context 2	51.95	94.76	67.11
CKY Gold tags	*91.05*	*99.71*	*95.18*
CKY Supertagger	68.80	93.81	79.38
LSTM Top-down	**81.72**	95.33	**88.01**

Table 6.11: Results of the parsers for impersonal verbs identification.

Chapter 7

Conclusions

This chapter presents the conclusions of our work. We present a summary of the research results described so far, and discuss some known limitations. Then we describe some avenues of research that might be interesting to follow in the future.

7.1 Research results

We set out this research with the objective of creating a statistical HPSG parser for Spanish. This was accomplished by completing a series of steps.

HPSG grammar for Spanish

We designed a HPSG grammar adapted to Spanish. This grammar is different than the previously existing Spanish HPSG grammar (SRG [Marimon et al., 2007]) in that our grammar is built with the explicit design objective of creating a statistical parser. Because of this, the rules in our grammar tend to be very broad and capture a great number of situations, and we leave the finer-grained task of distinguishing between good uses and bad uses of the language to the statistical modules. In particular, we needed to be able to correctly model the situations present in the corpus we would use (Spanish AnCora).

Compared to the original standard HPSG grammar, ours adds some features for Spanish like the use of clitic pronouns, and some particularities for modeling verb phrases. We also simplify some aspects like the use of semantic role labeling as semantic representation instead of the minimal recursion semantics approach, and the modeling of only one kind of non-local dependency.

The feature structure that represents the words in our grammar contains features for:

- Part of speech and morphological attributes of words.
- Syntactic combinatorial valence.

- Only one type of non-local dependency: relative clauses acting as noun phrase modifiers.
- Semantic argument structure.

The grammar uses thirteen very generic rules:

- Two rules for attaching a specifier (or subject) to the left or to the right of a head.
- Two rules for attaching a complement to the left or to the right of a head, plus a rule for attaching a semantic complement to the right of a head.
- Two rules for attaching a modifier to the left or to the right of a head.
- One rule for attaching a clitic pronoun to the left of a head.
- One rule for attaching a relative clause to the right of a head.
- Two rules for attaching a punctuation symbol to the left or to the right of a head.
- Two rules for binarizing coordinations.

Corpus of HPSG sentences

We transformed the AnCora corpus of Spanish sentences into our HPSG format. We first used an automated process that transversed the corpus simplifying complex structures and using heuristics to detect heads and rules to apply for each constituent. Then we analyzed the resulting sentences using our feature structure implementation of the grammar, creating a proper HPSG version of the corpus.

The final corpus contains 517,237 instances of words (tokens) in 17,348 sentences. These words correspond to 83,594 unique lexical entries, which are structured in 3,245 lexical frames. The corpus was split in standard training, development and test partitions of around 80%, 10% and 10%.

Parsers

We implemented several parsing algorithms for our grammar trained over the data from our corpus. The algorithms belong to these three categories:

- A baseline bottom-up strategy that compares consecutive pairs of words in a sentence until finding the pair that is most likely to form a constituent, and repeats the process recursively until forming the whole tree. The best performing of these baselines is based on a multilayer perceptron that uses two words of context to the left or to the right of the words being analyzed.

- A statistical CKY approach that uses the lexical entries encoded as supertags, the possibility of applying the rules can be inferred from the supertags, and we use a PCFG style statistical model for guiding the CKY algorithm. We also trained a supertagger for finding the most likely supertags given the words of the sentence, built using a LSTM neural network.

- A top-down approach that recursively analyzes a sequence of words and splits it in the most likely point to form appropriate constituents. After forming the full trees, we use a second process that predicts the semantic arguments. Both steps are performed by LSTM neural networks.

Evaluation

We evaluated the performance of our parsing strategies and compared them to some well established Spanish parsers over the test corpus. We found that the LSTM top-down strategy outperforms all the other baselines in terms of constituency (87.57 U-Cons, 82.06 L-Cons), dependency (91.32 U-Dep, 88.96 L-Dep) and semantic role labeling metrics (87.68 U-SRL, 80.66 L-SRL). However, we must take in consideration that the results of the external parser had to be post-processed to comply to our format, and this could have added some noise to the comparison. In terms of speed, the LSTM top-down approach is faster than than the CKY parser, but lags behind other parsers such as spaCy or UDPipe.

We also evaluated the parsers using a set of metrics designed to test their performance on some particular Spanish phenomena. In this case, we found out that the CKY parser was the best parser at detecting postponed subjects (79.92 F1). On the other hand, the LSTM top-down parser was the best at detecting (98.90 F1) and classifying (70.60 macro-F1) clitics, and also detecting cases of clitics reduplication (34.00 F1), although the general performance for this was quite low for all parsers. LSTM top-down is also the best parser at detecting (89.60 F1) and classifying (59.02 macro-F1) relative clauses that modify a noun phrase, identifying the referents of these constructions (72.96 F1), detecting chains of coordinations (65.36 F1), identifying verbal periphrases (93.08 F1) and detecting impersonal verbs (88.01 F1).

7.2 Known limitations

The parsers we built, especially the LSTM top-down parser, seem to have good performance for our test corpus. However, this only means it behaves well when compared against data similar to the one it was trained for. Further research is needed to know if these results generalize to other types of data, for example text that is not from news, and also what differences in performance might arise when comparing against different variants of Spanish.

One aspect that might hinder the performance of our parser in a real scenario is our modeling of null subjects. We transformed the null subject marks from

the original AnCora corpus and kept them in our HPSG format, so our parsers benefited from this information during training. Notice that, when used with plain text, these markers would not be present, so the parsing performance would probably be lower. One solution to this is to pre-process the sentences using a classifier that is able to accurately detect null subjects and introduce the markers before parsing (see [González and Martínez, 2018]).

Another limitation related to the input format of the parsers is our handling of out of vocabulary words. As mentioned in section 5.3.3, we use a collection of more than a million word embeddings of size 300, but this does not guarantee that all the words in any input sentence will have an embedding representation in our collection. We use a technique for alleviating this problem that involves creating proxy embeddings for unknown words based on morphological information (e.g. the embeddings for a proper noun of type location would be the same as the embedding for *"España"*, which is the most common location in the corpus). This implies that we should have morphological information available for the words in a sentence prior to the parsing process. This is no problem when comparing with our HPSG version of the AnCora corpus because it has all the information we need, but in other cases it would be necessary to pre-process the input using a morphosyntactic analyzer such as FreeLing [Padró and Stanilovsky, 2012] that gives enough information. We have not carried experiments to see to what extent the parsing performance might be affected by having substandard POS-tagging or morphological information in the input.

7.3 Future work

During the course of the project we found several directions in which we think this work could be improved, that were not explored due to time limitations.

Improvements in the parsing process

The LSTM top-down parsing process performs well in terms of global metrics, and according to our evaluation it outperforms the baselines we compared to and is on par with other Spanish parsers. However, if we compare against the current state of the art for English, the performance of our parser lies between 85% and 92% of the performance achieved for similar metrics. This seems to be the case for Spanish parsers in general, so there is clearly still room for improvement in this direction.

An aspect that could be greatly improved is the speed of the parsing process. We tried to unify the two-step process of top-down parsing followed by SRL classification into a single neural network architecture, but its performance was much lower (see section 5.5). We would need a new way of thinking the architecture in order to unify both steps without compromising performance, for example using an attention mechanism or a time delayed layer.

There are also many refinements that could be done to the CKY strategy. First of all, the supertagger performance has plenty of room for improvement,

but even with a perfect supertagger the performance would not get past the upper bound set by the CKY with gold tags parser. In order to really improve the CKY performance, we would need to make a more powerful statistical model that takes into account more information like the lexical entries, the saturated and pending features, and the partial derivation history at each point.

Finally, we could try to combine both approaches by incorporating subcategorization information in the top-down strategy. Knowing the subcategorization features could help the top-down approach handle some situations better, for example postponed subjects. In order to do this we could feed the LSTMs both the word embeddings and an encoding of the supertags for each word. This would require using the supertagger previous to applying the top-down strategy as well, so improving the supertagger performance becomes essential for this scenario.

Improvements in the grammar

As mentioned before, our grammar only models one type of long range dependency, which is the relative clause that modifies a noun phrase. However, there is information in AnCora that might allow us to incorporate knowledge on longer distance dependencies. We would need to make changes in the grammar adding a filler-gap mechanism that could leave any number of gaps in some parts of the structure that are filled with elements found in other parts of the tree.

Further research is needed to understand how this could be integrated to a top-down parsing strategy, or if a combination of top-down and bottom-up heuristics might be needed in order to add this new complexity.

As well as this, it would be interesting to further analyze the behavior of the agreement principle (as seen in section 4.7) and implement strategies to enable it at least in some contexts. This could imply separating the morphological features in different classes so that only some of them are used for agreement checks, and also refining the rules so that the checks are done only at the appropriate times.

Experiments with other grammars

We could analyze more thoroughly the existing HPSG grammar for Spanish (SRG [Marimon et al., 2007]) to find ways of combining our version with this other grammar. The aim of this would be to incorporate the fine-grained linguistic knowledge existing in SRG (for example the very detailed type hierarchy for lexical entries), while at the same time trying to keep the flexibility of our approach. It would also push our model into the right direction for using a more complex semantic representation scheme, such as minimal recursion semantics, although this would probably need manual annotation or data curation.

On the other hand, it would be very interesting to try our approach applied to other types of deep grammars. We could, for example, apply the LSTM top-down strategy to existing an existing CCG corpus (like CCGbank [Hockenmaier

and Steedman, 2007]) or design a similar transformation for generating a Spanish CCG treebank and apply the parsing strategy to this new corpus.

Experiments in other languages

Our grammar was designed with some Spanish particularities in mind, but we consider it flexible enough so that it can be ported to other languages, albeit with some modifications. We can think of using a combination of dependency and constituency corpora for the same language (similar to the idea in [Zhou and Zhao, 2019]) that could at the same time give us proper constituency trees and heads information, and use some semi-automatic process to transform these to HPSG format. Then we could try our parsing approaches on these languages.

Experiments using other domains

The corpus we used to train our parsers consists in text from news articles, so the performance of the parsing process might be higher for text from this domain than for others. It would be interesting to analyze how well the parser performs on other domains, and how easy it would be to adapt it if the performance is not good enough. One type of text we could try to analyze is the legal domain, which uses a particular lexicon and also particular syntax where very long sentences are usual, often including high levels of subsection nesting and even itemization within a sentence. These complexities make it a very challenging domain for parsers in general. Besides the performance of the parsing process, we could focus on some aspects like the use of SRL. Given the performance on SRL for our parser seems to be good, we could try to apply this to legal text or other domains and use it as a first step for deeper analyses, for example for information extraction.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Alonso, H. M. and Zeman, D. (2016). Universal Dependencies for the AnCora treebanks. *Procesamiento del Lenguaje Natural*, 57:91–98.

Ambati, B. R., Deoskar, T., and Steedman, M. (2016). Shift-reduce CCG parsing using neural network models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 447–453.

Aparicio, J., Taulé, M., and Martí, M. A. (2008). AnCora-Verb: A Lexical Resource for the Semantic Annotation of Corpora. In *LREC*.

Azzinnari, A. and Martínez, A. (2016). Representación de Palabras en Espacios de Vectores. Proyecto de grado, Universidad de la República, Uruguay.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract Meaning Representation for Sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186.

Batalla, J. A., Pujadas, E. C., and Mayor, A. (2005). TXALA un analizador libre de dependencias para el castellano. *Procesamiento del Lenguaje Natural*, 35.

Bender, E. M., Flickinger, D., and Oepen, S. (2002). The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *Proceedings of the 2002 workshop on Grammar engineering and evaluation-Volume 15*, pages 1–7. Association for Computational Linguistics.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Bond, F., Oepen, S., Siegel, M., Copestake, A., and Flickinger, D. (2005). Open source machine translation with DELPH-IN. *Open-Source Machine Translation: Workshop at MT Summit X*, pages 15–22.

Bonial, C., Babko-Malaya, O., Choi, J. D., Hwang, J., and Palmer, M. (2010). PropBank annotation guidelines. *Center for Computational Language and Education Research Institute of Cognitive Science University of Colorado at Boulder*.

Buchholz, S. and Marsi, E. (2006). CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the tenth conference on computational natural language learning*, pages 149–164. Association for Computational Linguistics.

Callmeier, U. (2000). PET–a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6(01):99–107.

Carpenter, B. (1992). *The logic of typed feature structures*. Cambridge University Press.

Carreras, X. and Màrquez, L. (2005). Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the ninth conference on computational natural language learning (CoNLL-2005)*, pages 152–164.

Castellón, I., Fernández-Montraveta, A., Vázquez, G., Alonso, L., and Capilla, J. (2006). The SENSEM corpus: a corpus annotated at the syntactic and semantic level. In *5th International Conference on Language Resources and Evaluation (LREC 2006)*. Citeseer.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*.

Che, W., Liu, Y., Wang, Y., Zheng, B., and Liu, T. (2018). Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pages 740–750.

Chiruzzo, L. (2015). Construcción de Recursos Lingüísticos para una Gramática HPSG para el Español. Tesis de maestría, Universidad de la República, Uruguay.

Chiruzzo, L. and Wonsever, D. (2016). Transforming the AnCora corpus to HPSG. In Arnold, D., Butt, M., Crysmann, B., King, T. H., and Müller, S., editors, *Proceedings of the Joint 2016 Conference on Head-driven Phrase Structure Grammar and Lexical Functional Grammar, Polish Academy of Sciences, Warsaw, Poland*, pages 182–193, Stanford, CA. CSLI Publications.

Chiruzzo, L. and Wonsever, D. (2018). Spanish HPSG Treebank based on the AnCora Corpus. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*.

Chiruzzo, L. and Wonsever, D. (2019a). Building a supertagger for Spanish HPSG. *Computer Speech & Language*, 54:44–60.

Chiruzzo, L. and Wonsever, D. (2019b). Syntactic Analysis and Semantic Role Labeling for Spanish using Neural Networks. In *CICLing 2019 - 20th International Conference on Computational Linguistics and Intelligent Text Processing*.

Chiruzzo, L. and Wonsever, D. (2020). Statistical Deep Parsing for Spanish Using Neural Networks. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 132–144, Online. Association for Computational Linguistics.

Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

Chomsky, N. (1970). Remarks on Nominalization. *Readings in English Transformational Grammar, Waltham (Mass.): Ginn*, pages 184–221.

Chomsky, N. (1995). Bare phrase structure. *Evolution and revolution in linguistic theory*, pages 51–109.

Clark, K., Luong, M.-T., Manning, C. D., and Le, Q. V. (2018). Semi-supervised sequence modeling with cross-view training. *arXiv preprint arXiv:1809.08370*.

Clark, S. and Hockenmaier, J. (2002). Evaluating a wide-coverage CCG parser. In *Proceedings of the LREC 2002 Beyond PARSEVAL workshop*, pages 60–66. Citeseer.

Cocke, J. (1969). *Programming languages and their compilers: Preliminary notes*. New York University.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics.

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

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167.

Copestake, A. (2002). *Implementing typed feature structure grammars*, volume 110. CSLI publications Stanford.

Copestake, A., Carroll, J., Malouf, R., and Oepen, S. (1999). The (new) LKB system. *Center for the Study of Language and Information, Stanford University*.

Copestake, A., Flickinger, D., Pollard, C., and Sag, I. A. (2005). Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3(2-3):281–332.

Copestake, A. A. and Flickinger, D. (2000). An Open Source Grammar Development Environment and Broad-coverage English Grammar Using HPSG. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000)*, pages 591–600.

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102. Citeseer.

Cowan, B. and Collins, M. (2005). Morphology and reranking for the statistical parsing of Spanish. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 795–802. Association for Computational Linguistics.

Cross, J. and Huang, L. (2016). Span-Based Constituency Parsing with a Structure-Label System and Provably Optimal Dynamic Oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11. Association for Computational Linguistics.

Curran, J. R., Clark, S., and Vadas, D. (2006). Multi-tagging for lexicalized-grammar parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 697–704. Association for Computational Linguistics.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Dalrymple, M. (2001). *Lexical functional grammar*. Brill.

Davidson, D. (1967). The logical form of action sentences. *Essays on Actions and Events*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Di Tullio, A. and Malcuori, M. (2012). *Gramática del español para maestros y profesores del Uruguay*. ANEP.

Dozat, T. and Manning, C. D. (2017). Deep biaffine attention for neural dependency parsing. In *ICLR 2017*.

Dozat, T., Qi, P., and Manning, C. D. (2017). Stanford's Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada. Association for Computational Linguistics.

Dridan, R. (2009). *Using lexical statistics to improve HPSG parsing*. PhD thesis, University of Saarland.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., and Smith, N. A. (2015). Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343. Association for Computational Linguistics.

Dyer, C., Kuncoro, A., Ballesteros, M., and Smith, N. A. (2016). Recurrent Neural Network Grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209. Association for Computational Linguistics.

Flickinger, D., Zhang, Y., and Kordoni, V. (2012). DeepBank. A dynamically annotated treebank of the Wall Street Journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*, pages 85–96.

Friedman, D., Kasai, J., McCoy, R. T., Frank, R., Davis, F., and Rambow, O. (2017). Linguistically Rich Vector Representations of Supertags for TAG Parsing. In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pages 122–131, Umeå, Sweden. Association for Computational Linguistics.

Gaddy, D., Stern, M., and Klein, D. (2018). What's Going On in Neural Constituency Parsers? An Analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010. Association for Computational Linguistics.

Ghosh, D., Guo, W., and Muresan, S. (2015). Sarcastic or not: Word embeddings to predict the literal or sarcastic meaning of words. In *proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1003–1012.

González, L. and Martínez, V. (2018). Detección de sujetos omitidos en el español. Proyecto de grado, Universidad de la República, Uruguay.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štepánek, J., Straňák, P., Surdeanu, M., Xue, N., and Zhang, Y. (2009). The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18. Association for Computational Linguistics.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hockenmaier, J. and Steedman, M. (2007). CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Ingelmo, J. L. H. (2002). Los verbos soportes: el verbo dar en español. *Léxico y Gramática. Colección: Linguas e Lingüística. Lugo: TrisTram*, pages 189–202.

Ivanova, A., Oepen, S., Dridan, R., Flickinger, D., Øvrelid, L., and Lapponi, E. (2016). On different approaches to syntactic analysis into bi-lexical dependencies: An empirical comparison of direct, PCFG-based, and HPSG-based parsers. *Journal of Language Modelling*, 4.

Joshi, A. K. (1985). Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? *Cambridge University Press*.

Joshi, A. K. and Schabes, Y. (1991). Tree-adjoining grammars and lexicalized grammars. *Technical Reports (CIS)*, page 445.

Joshi, A. K. and Schabes, Y. (1997). Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–123. Springer.

Joshi, A. K. and Srinivas, B. (1994). Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 154–160. Association for Computational Linguistics.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA.

Jurafsky, D. and Martin, J. H. (2014). *Speech and Language Processing (3rd Edition)*, volume 3. Pearson London.

Kasai, J., Frank, R., McCoy, R. T., Rambow, O., and Nasr, A. (2017). TAG Parsing with Neural Networks and Vector Representations of Supertags. In *Conference on Empirical Methods in Natural Language Processing*, pages 1712–1722.

Kasami, T. (1966). An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*.

Kingsbury, P. and Palmer, M. (2002). From TreeBank to PropBank. In *LREC*. Citeseer.

Kolachina, P., Bangalore, S., and Kolachina, S. (2011). Extracting LTAG grammars from a Spanish treebank. In *Proceedings of ICON-2011: 9th International Conference on Natural Language Processing. Macmillan Publishers, India*, pages 2–3.

Le Roux, J., Sagot, B., and Seddah, D. (2012). Statistical parsing of Spanish and data driven lemmatization. In *ACL 2012 Joint Workshop on Statistical Parsing and Semantic Processing of Morphologically Rich Languages (SP-Sem-MRL 2012)*, pages 6–pages.

Lewis, M. and Steedman, M. (2014). Improved CCG parsing with semi-supervised supertagging. *Transactions of the Association for Computational Linguistics*, 2:327–338.

Liu, J. and Zhang, Y. (2017). Encoder-Decoder Shift-Reduce Syntactic Parsing. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 105–114, Pisa, Italy. Association for Computational Linguistics.

Lloberes, M., Castellón, I., and Padró, L. (2010). Spanish FreeLing Dependency Grammar. In *LREC*, volume 10, pages 693–699.

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

Marimon, M. (2010). The Spanish Resource Grammar. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC*, pages 17–23, Valletta, Malta.

Marimon, M., Bel, N., Espeja, S., and Seghezzi, N. (2007). The Spanish Resource Grammar: pre-processing strategy and lexical acquisition. In *Proceedings of the Workshop on Deep Linguistic Processing*, pages 105–111. Association for Computational Linguistics.

Marimon, M., Bel, N., Fisas, B., Arias, B., Vázquez, S., Vivaldi, J., Morell, C., and Lorente, M. (2014a). The IULA Spanish LSP Treebank. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA).

Marimon, M., Bel, N., and Padró, L. (2014b). Automatic selection of HPSG-parsed sentences for Treebank construction. *Computational Linguistics*, 40(3):523–531.

Marimon, M., Fisas, B., Bel, N., Vivaldi, J., Torner, S., Lorente, M., Vázquez, S., and Villegas, M. (2012). The IULA Treebank. In *LREC*, pages 1920–1926.

Marimon Felipe, M. (2010). The Tibidabo Treebank. *Procesamiento del lenguaje natural, 2010, vol. 45, num. 1, p. 113-119.*

Martí, M. A., Taulé, M., Bertran, M., and Màrquez, L. (2007). Ancora: Multilingual and multilevel annotated corpora. *Universitat de Barcelona.*

Matsuzaki, T., Miyao, Y., and Tsujii, J. (2007). Efficient HPSG Parsing with Supertagging and CFG-Filtering. In *IJCAI*, pages 1671–1676.

McDonald, R., Lerman, K., and Pereira, F. (2006). Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 216–220.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill, Inc., USA, 1 edition.

Miyao, Y., Ninomiya, T., and Tsujii, J. (2005). Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Natural Language Processing–IJCNLP 2004*, pages 684–693. Springer.

Miyao, Y. and Tsujii, J. (2005). Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 83–90. Association for Computational Linguistics.

Montague, R. (1970). English as a formal language. *Bruno Visentini (ed.), Linguaggi nella societa e nella tecnica. Edizioni di Communita*, pages 188–221.

Mrini, K., Dernoncourt, F., Bui, T., Chang, W., and Nakashole, N. (2019). Rethinking self-attention: An interpretable self-attentive encoder-decoder parser. *arXiv preprint arXiv:1911.03875.*

Ninomiya, T., Matsuzaki, T., Tsuruoka, Y., Miyao, Y., and Tsujii, J. (2006). Extremely lexicalized models for accurate and fast HPSG parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 155–163. Association for Computational Linguistics.

Nivre, J. (2003). An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 149–160, Nancy, France.

Nivre, J., De Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., et al. (2016). Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

Nivre, J., Hall, J., Nilsson, J., Eryiğit, G., and Marinov, S. (2006). Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225.

Packard, W. (2013). ACE: the Answer Constraint Engine. *URL http://sweaglesw.org/linguistics/ace*.

Padró, L., Collado, M., Reese, S., Lloberes, M., Castellón, I., et al. (2010). Freeling 2.1: Five years of open-source language processing tools. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*.

Padró, L. and Stanilovsky, E. (2012). Freeling 3.0: Towards wider multilinguality. In *LREC2012*.

Parsons, T. (1990). *Events in the Semantics of English*, volume 334. MIT press Cambridge, MA.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Peris Morant, A. and Taulé Delor, M. (2011). AnCora-Nom: A Spanish lexicon of deverbal nominalizations. *Procesamiento del lenguaje natural, 2010, vol. 46, p. 11-18*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Pineda, L. and Meza, I. (2005). The Spanish pronominal clitic system. *Procesamiento del lenguaje natural*, 34:67–103.

Pollard, C. and Sag, I. A. (1994). *Head-driven Phrase Structure Grammar*. University of Chicago Press.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

Rello, L., Baeza-Yates, R., and Mitkov, R. (2012). Elliphant: Improved automatic detection of zero subjects and impersonal constructions in Spanish. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 706–715. Association for Computational Linguistics.

Rosá, A., Chiruzzo, L., Etcheverry, M., and Castro, S. (2017). RETUYT in TASS 2017: sentiment analysis for Spanish tweets using SVM and CNN. *arXiv preprint arXiv:1710.06393*.

Sag, I. A., Wasow, T., and Bender, E. M. (2003). *Syntactic theory: A formal introduction*. Center for the Study of Language and Information Stanford, CA, 2nd edition.

Shen, Y., Lin, Z., Jacob, A. P., Sordoni, A., Courville, A., and Bengio, Y. (2018). Straight to the Tree: Constituency Parsing with Neural Syntactic Distance. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1180, Melbourne, Australia. Association for Computational Linguistics.

Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1201–1211.

Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. (2011a). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136.

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011b). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 151–161.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical*

Methods in Natural Language Processing, pages 1631–1642. Association for Computational Linguistics.

Steedman, M. (1996). A very short introduction to CCG. *Unpublished paper. http://www.coqsci.ed.ac.uk/steedman/paper.html*.

Stern, M., Andreas, J., and Klein, D. (2017). A Minimal Span-Based Neural Constituency Parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827. Association for Computational Linguistics.

Straka, M. and Straková, J. (2017). Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.

Takaki, M., Minoru, Y., Kentaro, T., and Jun'ichi, T. (1998). LiLFeS: towards a practical HPSG parser. In *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, pages 807–811. Association for Computational Linguistics.

Taulé, M., Martí, M. A., and Recasens, M. (2008). AnCora: Multilevel Annotated Corpora for Catalan and Spanish. In *Lrec*.

Taylor, A., Marcus, M., and Santorini, B. (2003). The Penn treebank: an overview. In *Treebanks*, pages 5–22. Springer.

Vaswani, A., Bisk, Y., Sagae, K., and Musa, R. (2016). Supertagging with LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015). Grammar As a Foreign Language. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2773–2781, Cambridge, MA, USA. MIT Press.

Watanabe, T. and Sumita, E. (2015). Transition-based Neural Constituent Parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1169–1179. Association for Computational Linguistics.

Xu, W., Auli, M., and Clark, S. (2015). CCG supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 250–255.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208.

Zeman, D., Hajič, J., Popel, M., Potthast, M., Straka, M., Ginter, F., Nivre, J., and Petrov, S. (2018). CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zeman, D., Popel, M., Straka, M., Hajič, J., Nivre, J., Ginter, F., Luotolahti, J., Pyysalo, S., Petrov, S., Potthast, M., Tyers, F., Badmaeva, E., Gökırmak, M., Nedoluzhko, A., Cinková, S., Jan Hajič, j., Hlaváčová, J., Kettnerová, V., Urešová, Z., Kanerva, J., Ojala, S., Missilä, A., Manning, C., Schuster, S., Reddy, S., Taji, D., Habash, N., Leung, H., de Marneffe, M.-C., Sanguinetti, M., Simi, M., Kanayama, H., de Paiva, V., Droganova, K., Alonso, H. M., Çağrı Çöltekin, Sulubacak, U., Uszkoreit, H., Macketanz, V., Burchardt, A., Harris, K., Marheinecke, K., Rehm, G., Kayadelen, T., Attia, M., Elkahky, A., Yu, Z., Pitler, E., Lertpradit, S., a! nd Jesse Kirchner, M. M., Alcalde, H. F., Strnadová, J., Banerjee, E., Manurung, R., Stella, A., Shimada, A., Kwak, S., Mendonça, G., Lando, T., Nitisaroj, R., and Li, J. (2017). CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhang, Y., Li, Z., and Zhang, M. (2020). Efficient Second-Order TreeCRF for Neural Dependency Parsing. *arXiv preprint arXiv:2005.00975*.

Zhang, Y., Matsuzaki, T., and Tsujii, J. (2010). Forest-guided supertagger training. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1281–1289. Association for Computational Linguistics.

Zhou, J. and Zhao, H. (2019). Head-driven Phrase Structure Grammar Parsing on Penn Treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.

Appendices

Appendix A

Deep Linguistic Formalisms

Section 2.1.4 introduced the idea of deep grammar formalisms, mentioned some characteristics they have, and gave some examples of formalisms that throughout the years have been considered deep grammars. Although a deeper analysis of these formalisms and their characteristics is out of scope of this work, in this appendix we will give a brief sketch of two of these formalisms.

A.1 Combinatory Categorial Grammars

Combinatory Categorial Grammars [Steedman, 1996] are grammars that build complex syntactic categories starting from simpler categories, and using operators to combine these so they can be applied to the left or to the right of other categories. Each lexical entry can be represented with one or more categories, and the tree derivations use only a few rules that indicate how the different categories can be combined. The main components of a CCG grammar are: the atomic categories, the complex categories (that are derived from the atomic ones) and the combination rules.

Suppose we want to build a CCG grammar for analyzing sentence samples 1, 2 and 9. The parts of speech in these sentences are the following:

- La gata duerme
  Det Noun Verb
- La gata negra duerme
  Det Noun Adj Verb
- La gata come el pescado
  Det Noun Verb Det Noun

We must first define the set of atomic categories for our grammar. In CCG very few basic categories are generally used. For our three sentences, one basic category for noun (N), one for noun phrase (NP), and one for the final sentence (S) are enough.

The complex categories are derived from the atomic categories in the following way:

- If X and Y are categories, then X/Y is a category.

  This category is the base for a forward application, and it represents an element that expects something of type Y on its right, and the result will yield something of type X.

- If X and Y are categories, then $X\backslash Y$ is a category.

  This category is the base for a backward application, and it represents an element that expects something of type Y on its left, and the result will yield something of type X.

These rules for building categories can be nested so that any number of forward and backward applications can be combined in a category. With this in mind, the rest of the categories for our example will be derived from the atomic categories (N, NP, and S) in the following way:

- A determiner will take a noun on its right and return a noun phrase: NP/N
- An adjective will take a noun on its left and return a noun: $N\backslash N$
- An intransitive verb will take a noun phrase on its left (its subject) and return a sentence: $S\backslash NP$
- A transitive verb will take a noun phrase on its right (its complement) and return something that must act just like an intransitive verb: $(S\backslash NP)/NP$

Now we can assign the corresponding categories to the words of sample 1 and make the CCG derivation. Knowing the categories, the derivation for this example is straightforward: first apply the noun to the right of the determiner to form a noun phrase, then apply the newly formed noun phrase to the left of the verb to form the sentence. This is shown below:

$$\begin{array}{ccc} \text{La} & \text{gata} & \text{duerme} \\ NP/N & N & S\backslash NP \\ \hline & NP & \\ \hline & S & \end{array}$$

The second example (sentence 2) is very similar, but we include an extra step for reducing the adjective with its noun on the left first. Then the derivation proceeds as the previous one, because the remaining categories are the same:

$$\begin{array}{cccc} \text{La} & \text{gata} & \text{negra} & \text{duerme} \\ NP/N & N & N\backslash N & S\backslash NP \\ & \multicolumn{2}{c}{\hrulefill} & \\ & N & & \\ \hline & NP & & \\ \hline & S & & \end{array}$$

Notice that these derivations look almost as an upside-down version of the trees in figure 2.1, which correspond to the same sentences. This is generally the case, as the derivation is in fact identifying the same constituents that could be captured using a CFG.

The main difference in the third example is that we must first consume the noun phrase on the right of the verb. Once that rule application is done, the remaining category is the same as the one used in the previous examples, so the derivation will go on in the same way:

La	gata	come	el	pescado
NP/N	N	$(S\backslash NP)/NP$	NP/N	N
NP			NP	
		$S\backslash NP$		
		S		

The two combination rules we used in these three derivations are the forward application (reducing X/Y Y to X) and the backward application (reducing Y $X\backslash Y$ to X). For these simple examples these rules are enough, but there are some more rules for dealing with more complex constructions, for example the composition rules, the type raising rule, and the coordinations rule. CCG are particularly good for deriving coordinations of complex semi-defined structures elegantly.

A.2 Tree Adjoining Grammars

Tree Adjoining Grammars [Joshi, 1985] are grammars built on the notion of tree substitution. The basic structures of the grammar are two types of trees: initial trees represent simple structures, while auxiliary trees allow recursion over the structures. These trees are combined through substitution and adjunction rules [Joshi and Schabes, 1997].

Everything in TAG is represented through a tree: the words, the sentences, and even the derivation process. The lexical entries for words like nouns and verbs will usually be initial trees, while the ones for adverbs and adjectives could be represented by auxiliary trees.

Suppose we want to build a tree for sentence 2 *"La gata negra duerme"*. First we must define the initial and auxiliary trees that will represent the different words. Figure A.1 shows these trees.

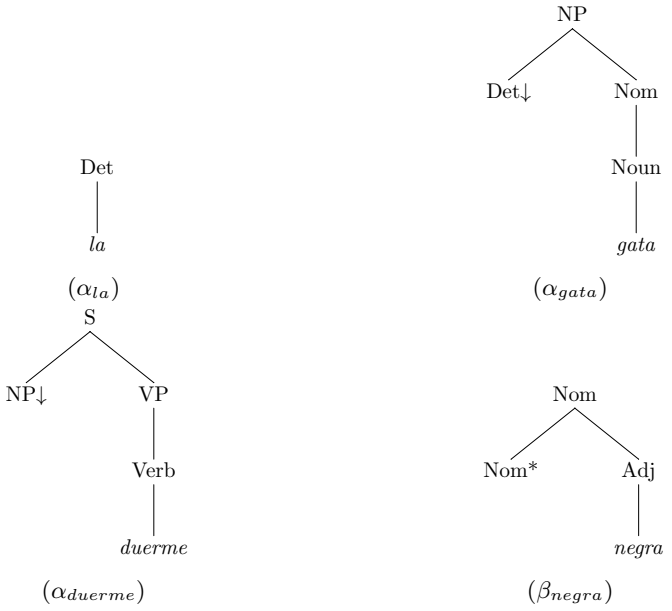


Figure A.1: TAG initial trees for *"la"*, *"gata"* and *"duerme"*, and auxiliary tree for *"negra"*.

Initial trees can contain underspecified nodes (marked with ↓) that indicate points where they expect a substitution with another tree. This can be seen in trees α_{duerme} and α_{gata}.

At the same time, an auxiliary tree must contain an underspecified node (marked with *) of the same type as its root node, as can be seen in tree β_{negra}. An auxiliary tree β marked with type X can be adjoined to another tree that has an element of type X and the whole tree β will be inserted in the parent tree in the corresponding position. This allows to add an arbitrary number of

subtrees to a derivation, which is especially useful for modeling the behavior of modifiers and other structures.

We can use these lexical entries to build a derivation of sentence 2. TAG makes the distinction between the derived tree (i.e. the final tree that results in the representation of the sentence) and the derivation tree (i.e. the tree that represents the derivation process) [Joshi and Schabes, 1991]. Figure A.2 shows a step by step derivation for sentence 2, showing which substitution or adjunction is performed at each step.

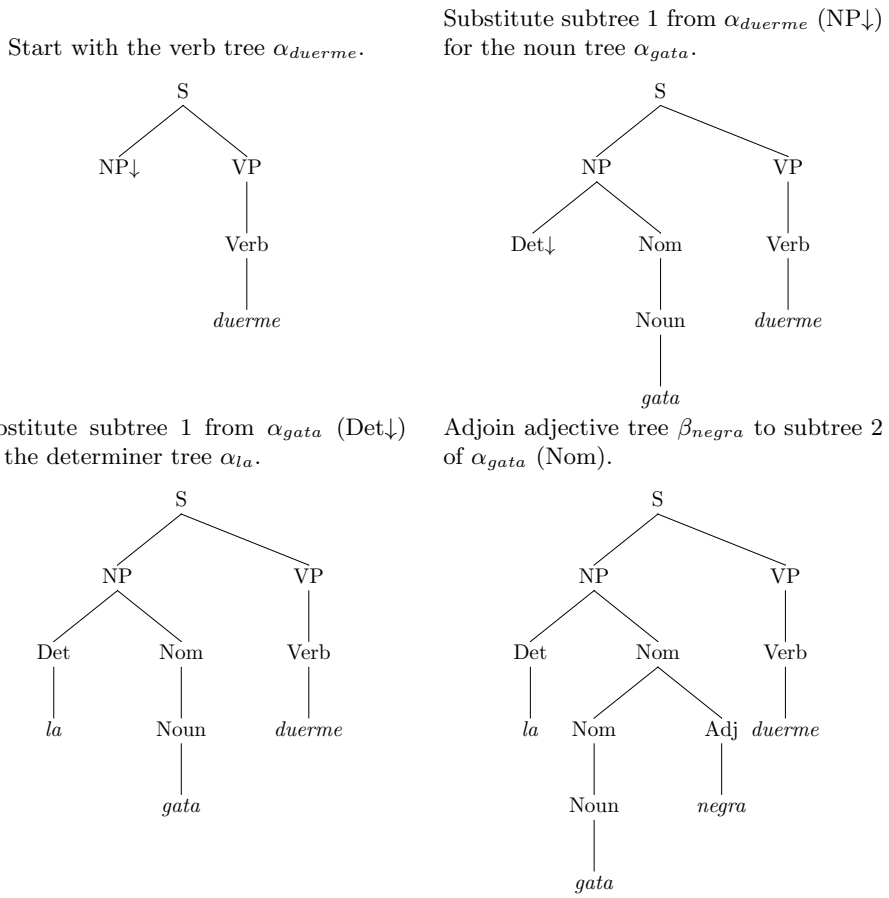


Figure A.2: TAG derivation of *"La gata negra duerme"*.

Notice that each substitution or adjunction operation is defined with respect to the trees introduced in the previous steps. When we want to specify a subtree inside a tree for an operation, we start counting from the left. For example, subtree 1 of α_{gata} represents the underspecified determiner Det↓, while subtree 2 of α_{gata} represents the Nom structure. This set of substitutions and adjunctions can also be described in a tree-like representation, called the derivation tree, as

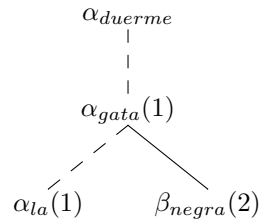


Figure A.3: TAG derivation tree for *"La gata negra duerme"*.

shown in figure A.3. In this notation, substitutions are shown as an arc between the parent tree and the tree being inserted marked with a dashed line, while adjunctions are shown as an arc marked with a solid line. The substituion or adjunction position is shown between parenthesis.

The final result of a derivation generally look very similar to a tree built using a CFG-style notation, but the derivation history also contains important information for understanding the sentence structure.

Appendix B

Parsing algorithms

In section 2.2 we presented one of the most important classical algorithms for parsing CFGs, the CKY algorithm. However, there exist other parsing algorithms, and some of the systems we discussed use variants of them. In this appendix, we will present another classical algorithm for parsing CFGs and one of the most widely used algorithms for parsing dependencies.

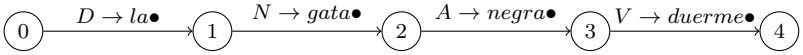
B.1 Chart parser

Chart parser is another dynamic programming parsing algorithm for CFG that can be adapted to be either top-down or bottom-up depending on the implemented strategy. This algorithm does not need the grammar to be in CNF. In a way it can be seen as a generalization of the CKY algorithm, only that instead of a matrix it will fill a chart with partial derivations of trees at each step.

Suppose we want to parse sentence 2 *"la gata negra duerme"* using the following grammar from section 2.1.1:

- $R_g = \{S \rightarrow NP\ VP,\ NP \rightarrow Det\ Noun,\ NP \rightarrow Det\ Noun\ Adj,\ VP \rightarrow Verb\}$
- $R_v = \{Det \rightarrow la,\ Noun \rightarrow gata,\ Verb \rightarrow duerme,\ Adj \rightarrow negra\}$

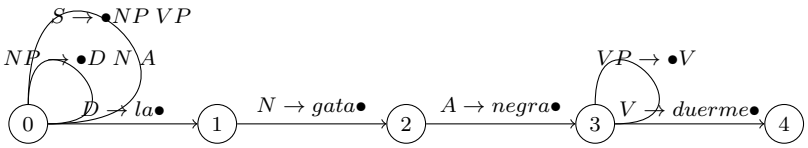
The algorithm starts with a graph containing one node for each position between words. The words will be represented as arcs between the corresponding position nodes.



Notice that each arc has a rule and a • sign. All arcs will have one of these signs at some position. The sign indicates the completion point for that rule. The arc labels will contain a rule annotated with a completion sign. For example, for rule $S \rightarrow NP\ VP$ we have the following possible labels:

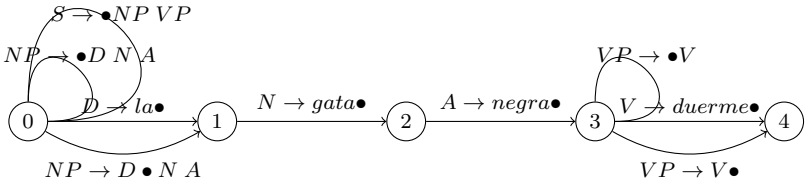
- $S \rightarrow \bullet NP\ VP$ indicates that for parsing a S, a NP followed by a VP are expected. This is an *active* label.
- $S \rightarrow NP \bullet VP$ indicates that a NP has been identified, and a VP is expected to form a S. This is also an *active* label.
- $S \rightarrow NP\ VP\bullet$ indicates that a NP followed by a VP have been identified, so the sequence adorned by this label is a complete S. This is an *completed* label.

Continuing with our example, we will add three active arcs to the graph that will help us parse the sentence: starting from node 0, one arc for forming a noun phrase, and another one for forming a sentence; and starting from node 3, an arc for forming a verb phrase.

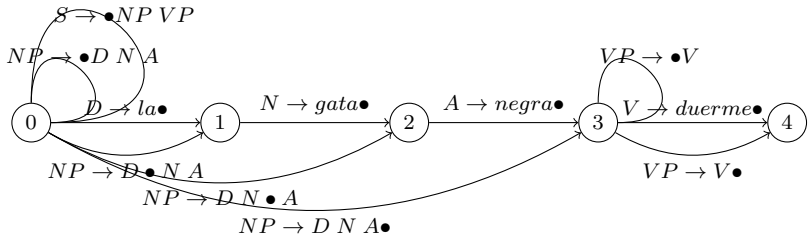


The fundamental rule of chart parsing states that, if we find an active arc from i to j of the form $A \rightarrow X \bullet BY$ and a completed arc from j to k of the form $B \rightarrow Z\bullet$, then we can add a new arc from i to k of the form $A \rightarrow XB \bullet Y$. Notice that X and Y might be empty. If Y is empty, the newly added arc would be complete.

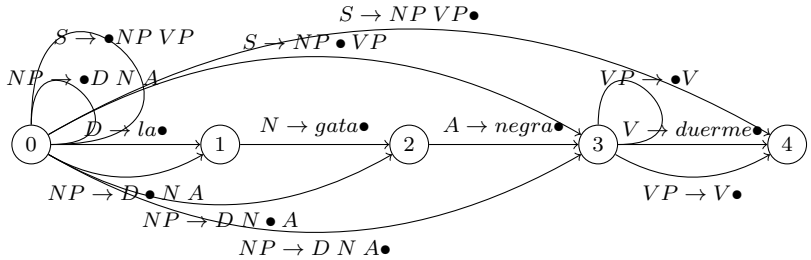
In our example, we can use the $NP \rightarrow \bullet D\ N\ A$ arc from 0 to 0 and the $D \rightarrow la\bullet$ arc from 0 to 1. We can also use the $VP \rightarrow \bullet V$ arc from 3 to 3 and complete it with the $V \rightarrow duerme\bullet$ arc from 3 to 4.



Now we will forward two steps: First combine $NP \rightarrow D \bullet N\ A$ from 0 to 1 with $N \rightarrow gata\bullet$ from 1 to 2. Then the result will be combined with $A \rightarrow negra\bullet$ from 2 to 3.



We have just completed a NP which can be used for the first part of our S, and we also have the VP that is needed to complete the whole S. We will combine $S \to \bullet NP\ VP$ from 0 to 0 with $NP \to D\ N\ A\bullet$ from 0 to 3. Then we will combine the result with $VP \to V\bullet$ from 3 to 4.



After this final step is complete, the algorithm has recognized the sentence using the whole sequence from nodes 0 to 4.

Notice that we arbitrarily chose to add those arcs in the second step because they would help us complete this particular parse. In a real scenario, there are strategies for adding the different arcs that could lead to find the correct parse tree more quickly or more slowly. Defining different rules for analyzing the agenda of pending constituents might derive in creating top-down, bottom-up, or mixed approaches based on different criteria.

As is the case with CKY, chart parsing is able to handle parsing ambiguity by having multiple possible arcs in the chart that generate the same sequences of words. This parser can also be used for grammars other than CFG, for example it has been applied to HPSG in the LKB framework (see section 2.3.1).

B.2 Transition-based dependency parsing

The algorithms shown in section 2.2.1 and appendix B.1 focus on using a CFG, or they can be adapted to other formalisms. Now we will present a widely used algorithm for dependency parsing that also can be adapted to other formalisms (including constituency parsing).

The transition-based dependency parsing algorithm [Covington, 2001; Nivre, 2003] transverses the sentence from left to right and makes decisions for each word. It keeps a stack of words that are being processed. At each step, the algorithm looks at the current word and the stack and takes one of three possible actions:

- `SHIFT`: Shift the current word to the top of a stack.
- `LEFT_ARC`: Create an arc between the two words on the top of the stack, so that the top word points at the previous word, then pops the previous word.
- `RIGHT_ARC`: Create an arc between the two words on the top of the stack, so that the top word is pointed by the previous word, then pops the top word.

Suppose we want to parse sentence 9 *"La gata come el pescado"* (*"The cat eats the fish"*) using this algorithm. Table B.1 shows the sequence of actions that this algorithm could take in order to parse the sentence. The result is a set of (unlabeled) dependencies between the words, which effectively describes the dependency parse tree shown in figure 2.3b. For simplicity, we are only considering the actions that correspond to making the unlabeled tree, but the approach can be expanded so that the actions `LEFT_ARC` and `RIGHT_ARC` also output a label for the dependency.

Stack	Words	Action	Dependency
[ROOT]	[la, gata, come, el, pescado]	SHIFT	
[ROOT, la]	[gata, come, el, pescado]	SHIFT	
[ROOT, la, gata]	[come, el, pescado]	LEFT_ARC	{ la ← gata }
[ROOT, gata]	[come, el, pescado]	SHIFT	
[ROOT, gata, come]	[el, pescado]	LEFT_ARC	{ gata ← come }
[ROOT, come]	[el, pescado]	SHIFT	
[ROOT, come, el]	[pescado]	SHIFT	
[ROOT, come, el, pescado]	[ ]	LEFT_ARC	{ el ← pescado }
[ROOT, come, pescado]	[ ]	RIGHT_ARC	{ come → pescado }
[ROOT, come]	[ ]	RIGHT_ARC	{ ROOT → come }

Table B.1: Example of transition-based dependency parsing for the sentence *"La gata come el pescado"* (*"The cat eats the fish"*).

We have not yet discussed how we decide which action to take in each step. The algorithm proposes that there is an *oracle*, a module that looks at the current state of the parser (for example the top two words of the stack and the

current word to process in the sequence) and decides which action to take. The important matter in this algorithm is how to build this oracle.

If we have treebank (see section 2.4), or a collection of previously analyzed sentences, we can build the parsing histories for each sentence, i.e. the sequence of actions that an oracle should take in order to find the correct parse. Then we can analyze the collection of parsing histories and create an oracle that outputs the action that is the most likely to be correct, for example using machine learning methods (see section 2.5).

This simplified version of the algorithm works in linear time on the length of the sentence. However, the actual execution order will depend on the complexity of the oracle used. If the oracle uses a lot of information from the stack, the sequence, or other inputs, the execution time will certainly increase.

Notice that the algorithm described so far is completely greedy, once it makes a decision there is no going back to fix it if something was wrong. This also means that there is no way to handle the ambiguity of a sentence using this algorithm, as it will only find one possible analysis. This is not necessarily a drawback, since this type of algorithms are designed to find the most likely tree, and they rely strongly on having a good statistical model that will predict the correct analysis in most cases. There could be extensions to this algorithm that incorporate the notion of backtracking on some decisions in order to cope with ambiguities or for searching the best solution more exhaustively.

By the nature of this algorithm, it is only able to find projective trees. There are extensions to this algorithm and also other dependency parsing algorithms (for example maximum spanning tree algorithm) that can deal with non-projective trees, although they are generally slower in execution time.

Appendix C

Glossary

accuracy Metric that measures the ratio of values correctly labeled by the classifier with respect to the whole data set. 51

activation function Function that computes the output of a neuron based on its inputs and weights. 52

agent Semantic role that indicates the participant that causes the action. 26

agreement Linguistic phenomenon present in some languages such as English and Spanish in which some of the words within constituents must have matching morphological characteristics. 12

agreement principle HPSG principle that tries to model the linguistic phenomenon of agreement. 24

ambiguity Property of sentences (or other linguistics constructions) that could be interpreted in more than one way. 14

annotated corpus Corpus with (manual) annotations used for building and evaluating statistical models. 48

attribute value matrix Representation of feature structures arranged as a (possibly nested) matrix of feature names and their values. 65

backpropagation Technique for updating the weights in a neural network that uses the difference between the output and the expected value and calculates the partial derivatives at each point propagating this information back through the network. 53

bi-directional LSTM A recurrent neural network layer composed of two LSTM layers: one reads the input from left to right and another one reads the input from right to left, then it concatenates the outputs. 56

binarization Way of modeling grammars in which each rule can only have two children elements. 13

candidate Data sample whose annotations are the predictions (outputs) of some statistical model. 50

clitic reduplication Phenomenon by which both a clitic pronoun and the argument it represents are present at the same time in the sentence. 81

coindexation Two features in a feature structure that point to the same value are said to be coindexed. 22

compositional semantics Approach to semantics in which the meaning representation of a larger unit is built by combining the meaning representations of its parts. 26

confusion matrix Matrix that contains the counts of expected values against values output by a classifier for each label. 50

content verb In a verbal expression, the verb in non-finite form that carries the main semantic content. 76

coordination chain Sequence of two or more elements that form a coordination, separated by other elements like conjunctions or punctuation symbols. 92

corpus A collection of sentences that is used as a representation of the language. 5

cross validation Technique for model selection and hyperparameter tuning that entails subdividing the training corpus in smaller subsets, training several models with these subsets, and comparing the average performance for those models. 50

dense layer A neural network layer composed of fully connected units. 54

dependency A relation between a pair of words in a sentence. 18

dependency tree Directed acyclic graph that represents that set of dependencies in a sentence. 18

dependent Word or other element that modifies or complements the head in a construction (for example in a phrase or a dependency relation). 18

development corpus Subset of a corpus that is used to evaluate and compare intermediate statistical models for model selection and hyperparameter tuning. 49

ditransitive verb A verb that takes a subject, a direct object and an indirect object as arguments. 13

dropout Technique that randomly selects at each training step a set of units that will not be used to calculate weights and will not be updated, used to prevent overfitting. 54

early stopping A criterion for stopping the training when the performance of the network over held-out data has not improved after some iterations in order to prevent overfitting. 54

elementary HPSG tree A structure composed of a head surrounded by its direct dependents (complements, specifier, modifiers). 98

embeddings layer A layer in a neural network that transforms the inputs into word embeddings representations. 57

endocentric feature A feature from a head that points to a dependent structure outside the head. 22

exocentric feature A feature from a dependent that points to the head of the construction. 22

expression Feature structure that represents the super-class of words and phrases. 66

F1 score Metric calculated as the harmonic mean between precision and recall. 51

feature structure An association of features and values, used in HPSG to represent words, phrases, rules, and trees themselves. 20

gold standard Data samples that are assumed to be correctly annotated. 50

grammar A set of rules that govern how to build sentences. 5

head Word that determines the syntactic properties of a construction (for example in a phrase or a dependency relation), and could sometimes stand alone for the whole construction. 18

held-out corpus Subset of a corpus used by some machine learning methods for tuning inner parameters of the method. 50

HPSG Head-driven Phrase Structure Grammar, a rich linguistic formalism that combines syntactic and semantic information in its analyses and is able to model many interesting linguistic phenomena. 5

hyperparameter Parameter (different than the input data) that modify the training process and which should be tuned to get optimal performance. 49

intransitive verb A verb whose only argument is the subject. 13

layer Arrangement of a set of units in a neural network that can be characterized by its set of inputs, outputs and connections. 53

L-Cons Labeled version of the constituency metric, equivalent to the F1 score of finding the expected constituents of the sentence, using the rule name as label. 116

L-Dep Labeled version of the dependency metric, equivalent to the Labeled Attachment Score using the rule name as label. 118

L-SRL Labeled version of the SRL metric, equivalent to the F1 score of predicting the expected semantic dependencies together with their argument types. 120

lexicalized grammar A grammar that focuses on modeling words (lexical entries) and the interactions between them. 17

loss function Target function of the optimization process when training a neural network, generally related to the difference between the output of the network and the expected output. 53

macro average Technique for averaging metrics in a multi-class classification problem, which assigns equal weight to all classes in the data set. 52

micro average Technique for averaging metrics in a multi-class classification problem, which assigns a weight to each class relative to the proportion of elements of that class in the data set. 52

non-compositional semantics Phenomenon in which the meaning of an expression cannot be built up from the meaning of its parts. 26

non-terminal Grammar symbol that can be rewritten as a combination of terminals or other non-terminals. They can be used for modeling constituents in a grammar. 10

one-hot encoding Input representation where the input vector has the size of the vocabulary, composed entirely of zeros except a value set to one in the position corresponding to the represented word. 55

overfit Phenomenon in which a statistical model has very high performance when evaluated against the data it was trained on but very low performance on the test data. 49

parser A system that performs syntactic analysis of sentences. 5

parse tree Representation of a sentence in a tree-like format, result of the parsing process. 9

parsing The process of taking a sentence and building a syntactic representation of it. 5

parsing history Sequence of steps followed to generate a parse tree from a sentence. 122

PP-attachment Prepositional phrase attachment, the problem of finding out the word or constituent a prepositional phrase should be attached to. 14

precision Metric that measures the proportion of times that the model was correct when classifying something as positive. 51

pre-terminal Non-terminal symbol that can only be rewritten as a terminal. They can be used for modeling parts of speech. 11

principle A strategy for sharing or percolating features from daughters to parent structures in a grammar. 20

projectivity Property of a dependency tree in which it is possible to draw the tree without crossing arcs. 19

quantifier scope ambiguity Ambiguity that happens when there are at least two possible interpretations for the scope of the quantifiers used in a sentence. 29

recall Metric that measures the proportion of positive values that were actually captured by the model. 51

relative pronominal expression Expression of one or more words that can introduce a relative sentence into another sentence. 85

root The word in a sentence that does not depend on any other word. 18

rule Description of a way for combining elements of a grammar to generate new elements. 10

saturated feature A feature that points to an empty list value, for example the `SPEC` or `COMP` features in a phrase when they are not expecting any more values. 24

semantic role A way of describing the semantic relation that a constituent of a sentence might have with respect to a predicate. 26

SRL Semantic Role Labeling, the task of assigning the correct semantic roles to a sentence. 27.

stochastic gradient descent Optimization method that implies starting with random weights and iteratively selecting a batch of samples, calculating the output for those samples, and adjusting the weights to make the actual output of the network more similar to the expected output, until some stop criteria is met. 53

subcategorization The capacity of verbs (or other words) to select the types of arguments they can be combined with. 13

supertag Fine grained label used to represent a complex category in deep grammars, for example a string representation of the lexical entry of a word. 35

supertagging The process of assigning the correct supertag to each word in a sentence. 35

support verb In a verbal expression, the conjugated verb whose number and person will agree with the subject. 76

terminal Grammar symbol that cannot be rewritten. They can be used for modeling words or other tokens in a grammar. 10

test corpus Subset of a corpus that is used to evaluate the final statistical models. 49

theme Semantic role that indicates the participant that suffers the effects of the action. 26

training The process of using a machine learning method to create a statistical model from a set of data. 48

training corpus Subset of a corpus that is used to train statistical models. 49

transitive verb A verb that takes a subject and a direct object as arguments. 13

treebank A corpus of sentences with their corresponding syntactic analyses. 17

typed feature structure Feature structures with an ontology-based type hierarchy definition for all the possible features and values. 20

U-Cons Unlabeled version of the constituency metric, equivalent to the F1 score of finding the expected constituents of the sentence, regardless of the label. 116

U-Dep Unlabeled version of the dependency metric, equivalent to the Unlabeled Attachment Score. 118

U-SRL Unlabeled version of the SRL metric, equivalent to the F1 score of predicting the expected semantic dependencies, regardless of the argument types. 120

unbalanced corpus Phenomenon in which there are too many examples of one class compared to the others in a data set. 51

underfit Phenomenon in which a statistical model has very low performance when evaluated against the data it was trained on. 49

unification Process by which the features of two feature structures are merged forming a new one containing the features of both, and goes on recursively unifying the values when the same feature is found in both structures. 20

word embeddings Technique that represents words in a dense vector space that is considerably smaller than the vocabulary size, hence embedding the language vocabulary into another structure while preserving its most salient features. 56